Section 3 Software Detail

Guide to the BIOS

Contents

Illustrations

Introduction

The BIOS is the Basic Input Output System of the Apricot. A part of the BIOS resides permanently in ROM and the other part is loaded into RAM when the system is powered up.

The ROM BIOS provides the primitives necessary for powering up the Apricot following switch on or a system reset. It contains the device drivers which control input and output to the hardware devices and diagnostics software.

The ROM BIOS also contains a Generic interface called the Control Device. This interface enables software to be written for the Apricot family even though Hardware Devices and their drivers within the ROM are totally different.

The RAM BIOS and other non-resident System files are loaded and initialised by the Bootstrap facilities within the ROM BIOS. The RAM BIOS is the interface between the operating system and the ROM BIOS.

A RAM BIOS is unique to the Operating System - in this case MS-DOS.

The following sections provide a guide to the BIOS and in particular to the following:

- the "Bootstrap" procedure
- the Label sector, system configuration data and minimal requirements of a Boot disk
- the initialisation of the BIOS and Device drivers
- the Memory map and reserved areas
- User interrupts

Bootstrap

The Bootstrap procedure is invoked by either a "cold start" (powering up the machine) or from a "warm start" by pressing the keyboard RESET button and holding it down for approximately 1 second.

A "cold start" performs diagnostic functions and a complete configuration of the Operating system.

A "warm start" assumes that certain data created by the "cold start" is intact but to ensure that this is reliable, it consults the "water mark" data in RAM for pre-defined values. The "water mark" is detailed in a later section on Pointers. The diagnostics are not run.

Following either start up, both the hardware devices whose drivers are resident in the ROM BIOS and their respective data areas in the RAM BIOS are initialised.

The initialisation of each driver is discussed in the next section.

The ROM BIOS then displays the start up screen which includes:

1. A logo.
2. The type of floppy disk drive as indicated by its disk capacity (normally 720 Kbytes).
3. The RAM size in Kbytes.
4. The capacity of Winchester drive where applicable (in MBytes).

If a diagnostics error occurs then a failure code as detailed in Appendix A, is displayed.

The ROM BIOS then initialises a 10 second timeout, displays a "Flashing hand" and key icons and waits for the operator to press the TIME/DATE key. Depression of the TIME/DATE key before the timeout elapses ensures that the system clock is updated from the battery backed-up clock in the keyboard.

On receipt of the TIME/DATE key, or on expiry of the timeout, the ROM BIOS commences searching for a Bootable disk. The Floppy drive is searched first. If it does not contain a suitable boot disk, and a bootable Winchester is present in the system, loading will be carried out from it instead.

Certain fields within the Label Sector of the disk identify a Bootable disk. The Label Sector is detailed in full at the end of the chapter. The following fields must, however, contain valid Bootstrap data:

LBL_Boot_Disk	offset 0BH	byte
LBL_winchester	offset 0DH	byte
LBL_boot_locn	offset 1AH	dword
LBL_boot_size	offset 1EH	word
LBL_boot_addr	offset 20H	dword
LBL_boot_st_off	offset 24H	word
LBL_boot_st_seg	offset 26H	word
LBL_BPBsctr_sz to LBL_BPBstart_sct	offset 50H to offset 5EH	16 bytes
CNF_ver_lo - thru CNF_nnn	offset 80H-FFH	128 bytes
If a Winchester disk:		
WINbad_sect_tab	offset 100H	32 words

If the disk is a valid bootable disk, the ROM BIOS will attempt to load the sectors indicated by LBL_boot_locn and LBL_boot_size from disk into RAM at the address given by LBL_boot_addr. Note that the code to be booted must be one contiguous block of sectors.

If all is well, the ROM BIOS loads up the internal configuration table from the boot disk label sector (128 bytes starting at CNF_ver_lo), clears interrupts and jumps to the location given by:

LBL_boot_st_seg:LBL_boot_st_off.

If, however, there is a failure, an error number (as detailed in Appendix A) is placed on the screen, and the ROM BIOS deselects the drive until a new disk is inserted.

Initialised drivers

The list of drivers given below are resident in the ROM BIOS and they are initialised by the Bootstrap procedure as follows:

Screen Driver

This clears the screen and places the cursor in the top left-hand corner of the screen. Screen output is to the LCD only, with colour output disabled.

Keyboard Driver

This clears out internal keyboard queues and status indicators, and all keyboard input is ignored except for the CALCULATOR, SET TIME, KEYBOARD LOCK, and TIME/DATE facilities.

Disk Driver

This clears internal disk statuses and then tests for a second disk drive by selecting it and checking for track 0. Note that the disk drives perform an automatic RESTORE on power-up, thus activating the TRACK00 signal. These tests take about 2.5 seconds and are only performed on a power-on reset.

Printer Driver

This clears the internal (2 Kbyte) printer buffer, and sets the data strobe output line to idle.

RS-232 Driver

This initialises the RS-232 driver to the internal configuration of 9600 baud, 8 bit per character, no parity, and 1.5 stop bits for transmission and reception.

Clock Driver

This resets the ROM BIOS clock to midnight on the 1st Jan 1980 (days = hours = mins = secs = hundredths secs = 0000.), and sets the BIOS clock running.

Winchester Driver

This checks for the presence of a Winchester controller board by testing for RAM in the Winchester buffer port. If RAM exists there, the ROM BIOS assumes the presence of a Winchester. The Winchester(s), if present are restored, and the label sector(s) read to check the size of the drive(s).

Initialised BIOS

When the ROM BIOS passes control to the operating system software loaded from disk, the machine is in the following state.

All 256 interrupt vectors are initialised, including all hardware and ACT defined interrupts. Unused vectors are set to point to a dummy handler in the ROM. Detection of a non-resident driver can therefore be made by checking if it's interrupt vectors point to an area above 0F8000 hex.

A list of Software interrupts and reserved interrupts is provided in a later section.

A pointer area as described in the section Memory Map below is also initialised.

A minimal set of 128 characters is provided in ROM for the Screen and a default Keyboard Table (see appendix B). In order to use a full set of characters, and to enable the reprogramming of keys, the RAM BIOS normally loads a new Font and keyboard into its own RAM space and changes pointers as appropriate, to point to the start of these new Font and Keyboard Tables.

The Font provided in the ROM BIOS is a standard 128 character ASCII set, implemented in an 8 x 8 matrix.

All hardware drivers are in an initialised state and can be called immediately. However, it is recommended that these drivers are initialised via the control device INIT calls (command code 0), except for the Clock Driver which should not need resetting, before the loaded operating system makes use of them.

Built-in functions

While the ROM BIOS is waiting for a boot disk the user can make full use of the following built-in facilities:

The Calculator

This appears on the 25th line of the screen, and is activated/deactivated by the CALC key (Shift + F4).

Keys used by the calculator are: Numeric Pad 0 - 9 and (.), ENTER, CLEAR, %, *, /, -, +, M+ (F3), M— (F8) and RECALL (F7). After boot the calculator also responds to the SEND (F9) key.

The SET TIME Key

At any time (before or after booting) the SET TIME key can be entered. This displays a prompt on the 25th line for time and date setting of the internal keyboard clock. The user must enter the date and time as a string of digits from the numeric keypad in the form HH MM DD MM YY (no spaces). The time is sent to the Systems Unit by pressing the TIME/DATE key. The screen prompt is removed if an invalid key is entered, more than 10 digits have been entered, or the TIME/DATE key has been pressed.

The TIME/DATE Key

This key can be pressed at any time, its function is to transmit the keyboard's internal time and date to the main unit, and is used during the boot process to start off the boot. During normal operation there is no visual indication of entry of this key - the ROM BIOS simply updates its internal clock with the data received.

Memory Map

The Memory Map below shows the initialised state of the Apricot after Bootstrap is complete.

The ROM Bootstrap initially loads the RAM BIOS Code and Data together with SYSINIT from the disk. SYSINIT is a module responsible for loading and initialising MS-DOS and is discarded after use.

SYSINIT is moved to a scratch area in high RAM and takes over the loading of the system. It loads the operating system MS-DOS and the system files containing the Keyboard Table and character Fonts.

The SYSINIT area is overwritten by MS-DOS.

The illustration shows two possible states, one where the Keyboard and Font files are not available and the other where they are.

To load a system without RAM based Font and/or keyboard table, the fields CNF_FONT_sec and/or CNF_KEY_sec in the boot disk label sector must be set to zero. In this case the ROM keyboard table and limited Font are used by the system.

Absolute addresses are given on the left of the diagram, space occupied in Kbytes on the right

The memory map shows both ROM and RAM areas. The area between F8000H and FFFFFH is ROM, and includes the ROM BIOS code, default keyboard table and character Font, and default BIOS constants.

The Keyboard and Font tables provide support for a default 128 character set. The Font is implemented in an 8 x 8 matrix.

The ROM BIOS constants are copied to RAM during the boot procedure, being amended there as necessary.

The area between 00000H and 05000H is used by the ROM BIOS, and its layout is fixed; it is termed 'ROM specified RAM'. It comprises the interrupt vectors, BIOS pointer area, ASCII screen images, and the ROM BIOS data, stack and configuration table.

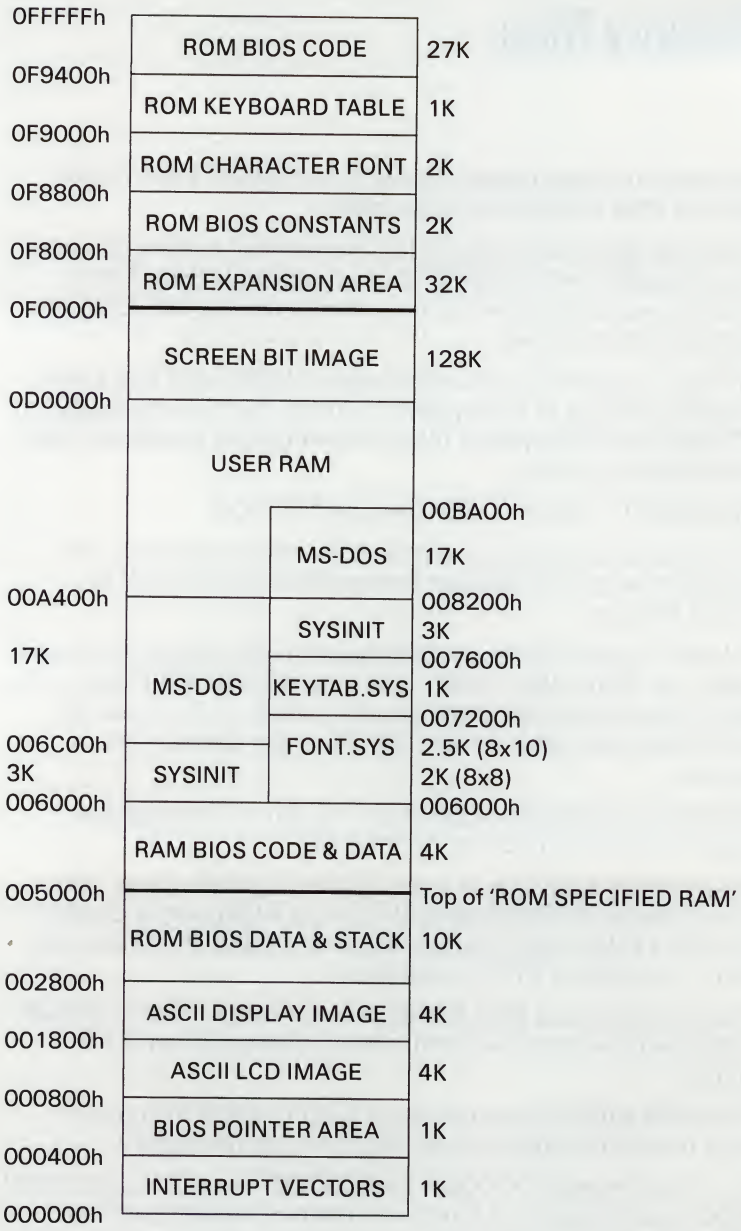


Figure 1. Apricot PORTABLE Memory Map

Software interrupts

The base page of the RAM is reserved for interrupt vectors. These are double word Segment:Offset pointers to the start address of the interrupt handler routine.

The pointers conform to Intel standard format, i.e.

<address> + 3: segment high
<address> + 2: segment low
<address> + 1: offset high
<address> + 0: offset low

The location of a pointer is determined by multiplying the interrupt number by 4; i.e. the base address of the interrupt F0 hex pointer is at location 4 x F0H or 03C0 hex.

A number of interrupts are specified as reserved by Intel and Microsoft. These are:

Intel: 0 - 1F hex
MS-DOS: 20 - 3F hex

Software interrupts are initialised to point to a dummy handler in the ROM BIOS. This handler simply executes an interrupt return (RETI).

Further interrupts are reserved by ACT for BIOS and Application use. These are listed below.

The BIOS initialises its software interrupt pointers by placing the address of appropriate routines in the respective interrupt vector location.

It is recommended that Application routines wishing to use a software interrupt observe the following rules:

1. Save the existing pointer (including the dummy pointer).
2. Store their "routine" pointer at the vector location.
3. Save all registers on entry.
4. Restore all registers before exiting the routine.
5. Perform a long jump to the "saved" pointer at the end of the routine. This ensures that any unknown routines will still be executed.

Note: The dummy routine is in ROM, i.e. all pointers to it have a segment address greater than or equal to F000 hex. This is useful for determining whether a software interrupt vector is in use. In turn this may also indicate whether a loadable device driver is present.

0FFH - Clock Interrupt every 20 ms

(see note 1)

Source: ROM BIOS
Input: none
Output: none

0FEH - Exec interrupt

Source: Application
Input: none
Output: AX = 0FFFFH

0FDH - Spooler Interrupt

Source: Application
Input: none
Output: AX = 0FFFFH

0FCH - BIOS Control Device interrupt

Source: RAM BIOS & Application
Input: AX, BX, CX, DX, SI
Output: AX - Other registers preserved.

0FBH - Mouse Interrupt 2

Source: Application
Input: none
Output: AX = 0FFFFH

0FAH - Mouse Interrupt 1: MicroSoft serial Mouse

(see note 1)

Source: ROM BIOS
Input: AL = Mouse Data Byte
Output: Not applicable.

0F9H - Keyboard Interrupt

(see note 2)

Source: ROM BIOS
Input: AL = BL = Decoded Key
AH = BH = Count of characters in string
Output: AX = 0FFFFH - pass key to DOS queue
AX = 00000H - Ignore key
BL = Translated Character

0F8H - Voice Interrupt 3

Source: Application
Input: none
Output: none

0F7H - Voice Interrupt 2

(see note 1)

Source: ROM BIOS
Input: AX = Voice Key Code Packet
Output: none

0F6H - Voice Interrupt 1

Source: Application
Input: none
Output: none

0F5H - Mouse Interrupt 3

(see note 2)

Source: ROM BIOS
Input:

AX	= Mouse packet
bits 0 - 7	= X or Y data
bit 8	= 1 - Y packet, 0 - X packet
bit 9	= Switch 1 (1 = on)
bit 10	= Switch 2 (1 = on)
bit 11	= 1
bits 12 - 15	= undefined

Output: None

0F4H - External Expansion Interrupt Setup

Source: Application
Input: AL = 0 - Set External Interrupt 2 (Winchester)
AL = 1 - Set External Interrupt 3 (general)
BX = Vector Offset Word
CX = Vector Segment Word
Output: BX = Old Vector Offset Word
CX = Old Vector Segment Word

0F3H - Voice Interrupt 4

Source: Application
Input: n/a
Output: n/a

0F2H - Special 25th line output

Source: ROM BIOS and Application
Input: CX = command
AL = data
Output: none, all registers preserved
Commands:
CX = 0, init. line 25. AL = ID code (see below)
CX = 1, reset line 25. AL = ID code (see below)
CX = 2, print character in AL
CX = 3, set cursor to position AL (0 - 79)
ID codes: 0 = voice, 1 = calc.

0F1H - Screen Output

(also provided by INT 29H - see note 3)

Source: RAM BIOS & Application
Input: AL = character
Output: none, all registers preserved.

0F0H - SIO Control Interrupt

(see note 1)

Source: ROM BIOS
Input: AL = SIO interrupt vector
0 = Ch B Tx buffer empty
2 = Ch B ext/status int.
4 = Ch B Rx ready
6 = Ch B Special Rx
8 = Ch A Tx buffer empty
10 = Ch A ext/status int.
12 = Ch A Rx ready
14 = Ch A Special Rx
Output: none

0EFH - Reserved for use by ACTEX

Redirected if INT 24 error handler present

0EEH - Send Non-specific end-of-interrupt (EOI) to PIC, or dummy 'RETI' sequence to Z80 chips

Source: Application Hardware Interrupt Handler
Input: none
Output: none

0E0H - GSX-86

Source: Application
Input: CX = 0473H - Function code
DS:DX - Parameter Block pointer
Refer to GSX chapter.

Note 1: SS, DS, ES must be preserved, the Stack must not be changed if interrupts are enabled, or a call to the ROM BIOS is to be made.

Note 2: All other registers must be preserved. The Stack must not be changed if interrupts are enabled, or a call to the ROM BIOS is to be made.

Note 3: Interrupt 29 hex is set up by the RAM BIOS for use by MS-DOS and can be re-used by other operating systems, since screen output is already provided by interrupt F1H. The ROM BIOS only uses interrupt F1H for screen output and never interrupt 29H.

Hardware interrupts

The following interrupts are assigned to hardware:

50H - Programmable Interval Timer (system clock)
51H - Floppy Disk Controller Interrupt
52H - Voice Controller interrupt
53H - External Interrupt 2 (Winchester)
54H - Z80 SIO interrupt
55H - External Interrupt 3
56H - Parallel Printer Interrupt
57H - DMA Controller Interrupt

Pointers

The ROM BIOS accesses a 1K byte Pointer area located between address 400H and 7FFH in the RAM.

The pointer area is initialised by the Boot procedure. It contains constants and double word pointers.

The Pointers provide the BIOS and Application with details of the Operating System as follows:

- version number
- Boot details and diagnostic results
- disk drives and their capacities
- table pointers and sizes
- memory map details

Certain pointers may be changed freely by the Applications, such as those that point to character Font's, but others must NOT be altered.

The Table given below details each pointer, it's size in bytes and an indication of whether it is freely modifiable. All data and pointers can of course be referenced by the Application.

Double-word pointers (four bytes) conform to the standard Intel addressing formats, i.e. the first two bytes are the offset within the segment, and the second two bytes are the segment address.

Example: Accessing the Pointer area from Basic:

```
10 DEF SEG=0
20 FONT=PEEK(&H0706) +256*PEEK(&H0707)
30 DEF SEG=PEEK(&0708)+256*PEEK(&H0709)
```

Statement 30 sets the variable FONT to the offset of the active character Font, and statement 30 sets the current segment for future Peek's to the segment containing the Font.

Address	Length (in bytes)	Set by	Use
400H	1	BOOT	Boot (P)ROM version number
401H	1	BOOT	Machine type 0 = Apricot & Apricot XI 1 = Apricot Portable 2 = Apricot F1 & F1e
402H	2	BOOT	RAM Memory Size (in paragraphs)
404H	4	BOOT	Cold/Warm bootstrap mark (5678H, 1234H)
408H	2	BOOT	Drive booted from 0000H = Floppy 0 0001H = Floppy 1 0002H = Winchester 1 0003H = Winchester 2
40AH	4	BOOT	Pointer to loaded boot disk header sector (immediately after Boot only)
40EH	4	BOOT	Reserved
412H	2	BOOT	Reserved
414H	1	BOOT	Winchester type: 0 = No Winchester 3 = 5 Megabyte RO351 Winchester 4 = 10 Megabyte RO352 Winchester 5 = 20 Megabyte Winchester
415H	1	BOOT	Floppy type: 0 = 70 track SS 1 = 80 track SS 2 = 80 track DS
416H	1	BOOT	Number of Floppy Drives
417H	1	BOOT	Number of Winchester Drives
418H	4	BOOT	Pointer to Floppy BPB array Table
41CH	4	BOOT	Pointer to Winchester BPB array Table

Address	Length (in bytes)	Set by	Use
420H	2	BOOT	Power-up Diagnostic Results Test failure number from boot diagnostic
422H	2	BOOT	Paragraph address of start of user code area (for use by BIOS's loaded from disk).
424H	4	BOOT	ROM BIOS Stack Segment/Offset (for use by interrupt routines)
500H	8		Reserved
508H	8		Reserved
600H	5	BOOT	Long jump to BIOS Control Device
610H	112	USER	Reserved for MBASIC/GSX interface
700H	4	BOOT	Pointer to internal BIOS config table
704H	2	BOOT	Length of internal BIOS config table in bytes
706H	4	BOOT/ USER	Pointer to active character Font
70AH	2	BOOT/ USER	Length of active Font in bytes
70CH	4	BOOT/ USER	Pointer to master character Font
710H	2	BOOT/ USER	Length of master Font in bytes
712H	4	BOOT/ USER	Pointer to active keyboard tables
716H	2	BOOT/ USER	Length of active keyboard tables
718H	4	BOOT	Pointer to internal keyboard tables
71CH	2	BOOT	Length of internal keyboard tables
71EH	4	BOOT/ USER	MicroScreen Character Table pointer. Not used.
722H	4	BOOT	Pointer to write only register copy table
726H	4	BOOT	Pointer to ASCII colour screen image

Address	Length (in bytes)	Set by	Use
72AH	4	BOOT	Pointer to ASCII LCD screen image
780H	4	USER	Voice software pointer area

ASCII and Bit Screen Images

Three areas of the RAM are reserved for the Displays.

The first area is the bit mapped Display RAM which is used by the driver software to place images on the screen (either the LCD or colour Display). This is called the Bit Image.

The other two areas are used by the BIOS, to maintain an ASCII image of the LCD and the Display. These images are called the Screen Images and are used to store the characters and their respective attributes. They provide the applications programmer an easy way of updating the bit-mapped displays. Apart from being used to update the Bit Image these areas are used to implement features such as Hardcopy.

The Bit Image is used for both LCD and Display and this is fully described in the Screen Driver chapter.

The Apricot has 128K bytes of addressable Bit Image. Two hardware configurations are available; the base model of the Apricot Portable range has a 64K Bit Image; the second model has the maximum of 128K bytes.

RAM BIOS for MS-DOS

The function of the RAM BIOS is to link MS-DOS to the ROM BIOS. It does this by passing MS-DOS calls to the Control Device; i.e. interrupt FCH.

Bootable Disks for MS-DOS 2.11 include a full generic RAM BIOS and is designed to boot on all Apricots equipped with a ROM BIOS.

The RAM BIOS will support any machine configuration up to 2 Floppy Drives and 2 Winchester Drives.

The Fixed Position Files on the Bootable Disk are as follows:

File	Start Sector (70T/ss)	(80T/ds)	Byte Length
FONT.SYS	0DH	12H	12.5K
KEYTAB.SYS	26H	2CH	1K
IO.SYS	28H	2EH	7K
MSDOS.SYS	36H	3CH	17K

Note: The size of MSDOS.SYS will increase for MS-DOS 3.00 to approximately 28K.

FONT.SYS contains a 16 x 16, an 8 x 8 and an 8 x 10 Font.

Disk Label Sector and Configuration Table

The Disk Label Sector for Generic Boot disks is situated in Track 0, Sector 0 on the boot disk (the first physical sector on the disk), and is 512 bytes long.

It contains both disk and operating system identification, boot loading information, and the ROM BIOS configuration table (this must be valid on the boot disk).

The first 80H bytes of the Label Sector contain disk and OS configuration data. The remainder, i.e. bytes 80H to FFH contains the configuration data for all other physical devices.

After the Disk Label Sector has been loaded by the Bootstrap, the address of it's location in RAM is placed in the Pointer area (see location 40A hex). The subsequent operating system loader and system set-up routines may then access the sector image as required. Following the initialisation of MS-DOS, the Sector image is not guaranteed, i.e. it is a facility supplied to support Booting and individual Systems must take responsibility for maintaining it's integrity.

The data in the Label Sector is also stored partly in the Pointer Table and the remainder in the Configuration Table.

A further pointer at 700H points to the internal copy of configuration parameters (2nd 80H bytes of the disk header).

First 80H bytes, disk & O/S identification

Offset	Reference	Bytes	Description
0000	LBLform_vers	8	version of format which created disk
0008	LBLop_sys	1	Operating System: 0 = invalid 1 = MS-DOS 2 = p-System 3 = CP/M 86 4 = Concurrent CP/M
0009	LBLsw_prot	1	software write protect (0 = off)
000A	LBLcopy_prot	1	copy protect (0 = off)
000B	LBLboot_disk	1	boot disk type: 0 = non-bootable disk 1 = Apricot & XI RAM BIOS 2 = GENERIC ROM BIOS 3 = Apricot & XI ROM BIOS 4 = Apricot Portable ROM BIOS 5 = Apricot F1 ROM BIOS
000C	LBLmulti_region	1	multi-regioned 0 = not multi-regioned nz0 = number of logical volumes)
000D	LBLwinchester	1	Winchester disk (1 = winchester)
000E	LBLSec_size		2sector size (in bytes)
0010	LBLsec_track	2	sectors per track
0012	LBLtracks_side	4	Tracks per side
0016	LBLsides	1	sides 1 = single 2 = double
0017	LBLinterleave	1	interleave factor
0018	LBLskew	2	skew factor
001A	LBLboot_locn	4	sector number of boot image
001E	LBLboot_size	2	number of bootstrap sectors
0020	LBLboot_addr	4	boot load address (dword pointer)

Offset	Reference	Bytes	Description
0024	LBLboot_st_off	2	boot start address offset
0026	LBLboot_st_seg	2	boot start address segment
0028	LBLdata_locn	4	sector number of first data block
002C	LBLgeneration	2	generation number
002E	LBLcopy_count	2	copy count
0030	LBLcopy_max	2	maximum number of copies
0032	LBLserial_id	8	serial number
003A	LBLpart_id	8	part number
0042	LBLcopyright	14	copyright notice
	Main Disk Extended BPB image:		
0050	LBLBPBsctr_sz	2	sector size in bytes
0052	LBLBPBclu_sz	1	cluster size in sectors
0053	LBLBPBrsvd_sct	2	reserved sectors
0055	LBLBPBn_fats	1	number of FAT's
0056	LBLBPBn_dir_ent	2	number of directory entries
0058	LBLBPBn_sectors	2	number of sectors
005A	LBLBPBmedia_id	1	media ID byte
005B	LBLBPBn_fat_sct	2	number of sectors per FAT
005D	LBLBPBdisk_type	1	type of disk: 0 = 70 track SS 1 = 80 track SS 2 = 80 track DS 3 = 5M winchester 4 = 10M winchester 5 = 20M winchester
005E	LBLBPBstart_sct	2	Logical sector for start of volume
	End of BPB image.		
0060	LBLfont_name	16	name of default FONT.SYS
0070	LBLkeys_name	16	name of default KEYTAB.SYS

Configuration constants definition area (Base address = Label Sector base + 80H)

Note: Run-time ROM BIOS pointer at 700H points to BIOS copy of these configuration tables.

Offset	Reference	Bytes	Description
			Systems unit:
0080	CNF_ver_lo	1	Disk BIOS minor version number
0081	CNF_ver_hi	1	Disk BIOS major version number
0082	CNF_diagflag	1	Global diagnostics flag (0 = off)
0083	CNF_lst_dev	1	PRN: device 0 = parallel 1 = serial
0084	CNF_Bell_vol	1	bell volume 0 = full 15 = off)
0085	CNF_cache_on	1	Reserved
0086	CNF_graphics_on	1	Reserved
0087	CNF_DOS_len	1	Length of DOS in sectors
0088	CNF_FONT_len	1	Length of FONT in sectors
0089	CNF_KEYS_len	1	Length of KEY table in sectors
008A	CNF_DOS_sec	2	Start sector of DOS image
008C	CNF_FONT_sec	2	Start sector of 3 Fonts (12.5K)
008E	CNF_KEYS_sec	2	Start sector of KEY table image

Note: The last six fields are private to the MS-DOS BIOS; other O/S's may use these for their own private uses. If FONT_sec or KEYS_sec are set to zero then the respective tables are not loaded, and BIOS uses the ROM versions.

Offset	Reference	Bytes	Description
			Keyboard:
0090	CNF_Click_vol	1	Key click volume 0 = full 15 = off
0091	CNF_rept_en	1	Auto-repeat master enable 0 = off 1 = on
0092	CNF_rept_dly	1	Auto-repeat lead-in (not used)
0093	CNF_rept_int	1	Auto-repeat interval (not used)
0094	CNF_Mscrn_mode	1	Microscreen mode (not used)
0095	11		spare
			Screen:
00A0	CNF_line_mode	1	0 = 256 line 1 = 200 line display
00A1	CNF_line_width	1	0 = 80 1 = 40 column display
00A2	CNF_image_off	1	0 = Screen Image on 1 = image off
00A3	13		spare

Offset	Reference	Bytes	Description
Serial communications:			
00B0	CNF_Tx_brate	1	Tx baud rate 1 = 50, 2 = 75, 3 = 110, 4 = 134.5, 5 = 150, 6 = 300, 7 = 600, 8 = 1200, 9 = 1800, 10 = 2400, 11 = 3600, 12 = 4800, 13 = 7200, 14 = 9600, 15 = 19200
00B1	CNF_Rx_brate	1	Rx baud rate 1 = 50, 2 = 75, 3 = 110, 4 = 134.5, 5 = 150, 6 = 300, 7 = 600, 8 = 1200, 9 = 1800, 10 = 2400, 11 = 3600, 12 = 4800, 13 = 7200, 14 = 9600, 15 = 19200
00B2	CNF_Tx_bits	1	Tx bits per char. (5 to 8)
00B3	CNF_Rx_bits	1	Rx bits per char. (5 to 8)
00B4	CNF_stop_bits	1	stop bits 1 = 1, 2 = 1.5, 3 = 2
00B5	CNF_parity_chk	1	parity check 0 = no check 1 = check
00B6	CNF_parity_typ	1	parity type 0 = none 1 = odd 2 = even 3 = mark 4 = space
00B7	CNF_Tx_xonxoff	1	transmit xon/xoff protocol 0 = off 1 = on
00B8	CNF_Rx_xonxoff	1	receive xon/xoff protocol 0 = off 1 = on
00B9	CNF_xon_char	1	XON character code
00BA	CNF_xoff_char	1	XOFF character code
00BB	CNF_Rx_X_limit	2	XON/XOFF receive buffer limit
00BD	CNF_dtr_dsr	1	DTR/DSR protocol 0 = off 1 = on

Offset	Reference	Bytes	Description
00BE	CNF_cts_rts	1	CTS/RTS protocol 0 = off 1 = on
00BF	CNF_CR_null	1	number of nulls to send after CR
00C0	CNF_FF_null	1	nulls (x10) to send after FF
00C1	CNF_s_cr_lf	1	Auto LF after CR 0 = off 1 = on
00C2	CNF_s_bioserr	1	BIOS error report 0 = off 1 = on
00C3		13	spare
Parallel communications			
00D0	CNF_p_cr_lf	1	auto LF after CR 0 = off 1 = on
00D1	CNF_select	1	select line support 0 = off 1 = on
00D2	CNF_pe	1	paper empty support 0 = off 1 = on
00D3	CNF_fault	1	fault line support 0 = off 1 = on
00D4	CNF_p_bioserr	1	BIOS error report 0 = off 1 = on
00D5		11	spare
Winchester:			
00E0		14	spare
00EE	CNF_wini_park	1	parking enable flag 0 = on nz = off
00EF	CNF_wini_form	1	format protection 0 = off nz = on

Offset	Reference	Bytes	Description
RAM disk:			
00F0		16	spare
Non-dedicated area of label sector starting at base + 100H; used for Winchester disk bad block tables and Multi-volume BPB's.			
0100	WINbad_sect	64	Up to 32 words giving logical sector numbers of bad blocks on the disk
0140	WINvol_bpb1	16	Extended BPB volume 1
0150	WINvol_bpb2	16	Extended BPB volume 2
0160	WINvol_bpb3	16	Extended BPB volume 3
0170	WINvol_bpb4	16	Extended BPB volume 4
0180	WINvol_bpb5	16	Extended BPB volume 5
0190	WINvol_bpb6	16	Extended BPB volume 6
01A0	WINvol_bpb7	16	Extended BPB volume 7
01B0	WINvol_bpb8	16	Extended BPB volume 8
01C0		63	spare
01FF		1	CP/M sides flag (0 = single sided)

Control device 3.2

Contents

Overview

The Control Device is a software driver which routes requests for I/O to the individual hardware drivers in the ROM BIOS. The philosophy behind this device is to provide a generic interface for the Apricot family.

Application software written using the Control Device will thus be compatible with future generations of the Apricot family.

Application software which detours the Control Device in order to access the hardware directly cannot assume compatability with future releases within the family. This practice is not to be condoned and should only be used in the last resort.

The Control Device is implemented fully on the F1 and Portable. At present only a subset is available on the pc and Xi but future versions will be brought into line with the inclusion of a full implementation of the generic Control Device.

Two methods of invoking the Control Device are provided to facilitate the needs of low and high level languages. These are described in detail in the next section.

An example of the low level requirement can be seen in the RAM BIOS which serves MS-DOS with all I/O functions. The RAM BIOS is implemented at low level machine code and it routes all I/O via the Control Device. High level languages such as BASIC may also make use of the Apricot features but do not have the same facilities to invoke the Control Device.

The following sections provide a blueprint for utilising the Control Device, a description of each of the hardware drivers together with examples of how to access them from a high level language.

Remember that the Control Device is a generic interface and that the inclusion of drivers in the following sections does not neccessarily imply that the hardware is available on a specific machine. The Microscreen is not available on the Portable for example but its driver is naturally a part of the generic family. Such occurences are noted and a description of the effect of calling a non-existant hardware module is included.

General application

Introduction

This section describes how to invoke the Control Device from low and high level languages.

The two methods each require the following arguments to be passed to and from the individual routines:

Entry:

P1 = Device number

P2 = Command

P3 = Data or Data pointer segment

P4 = Data or Data pointer offset

Exit:

PX = Data/Status

Reference to individual routines will determine the format of the parameters.

The first method is available at machine code level and is invoked by executing a software interrupt of type FC hex with the registers set up with the entry parameters.

The alternative method is available to facilitate languages which are not able to generate a software interrupt or indeed to access the processor registers.

This involves calling a fixed location in the Pointer area (0600 hex) which contains a "Long jump" to the Control Device interface routine with the parameters on the Stack. In Basic, for example, this is achieved with a CALL Statement.

Note that interpretive BASIC is used in this chapter and others purely as a universal quick reference to the use of facilities. It is not really a suitable language for accessing the Control Device and in certain cases still requires machine code subroutines to succeed. It is recognised that use of the Control Device will fall mainly into the category of the professional Application package developer using assembler, C, PASCAL, Compiled BASIC, etc.

Low level Control Device access

Entry:

Set the internal registers with the following "word" values:

BX = Device number (P1)

CX = Command (P2)

DX = Data or Data ptr segment (P3)

SI = Data or Data ptr offset (P4)

Call:

Execute software interrupt type FC hex

Exit:

AX = Data or Status (PX)

All routines in the Control Device preserve the entry registers. The AX register will not normally be preserved.

The above parameters and result are all "word" values. To facilitate functions which require additional data the DX register is combined with the SI register to provide a Segment:Offset pointer.

DX:SI = Segment:Offset

In this case the return data/status will normally be at the location pointed to by DX:SI.

The Device numbers are detailed in later sections. So too are the command values together with detail which will determine which routines require the "Data pointer" option and how each routine returns Data or Status.

High level Control Device access

Entry:

Set the following "word" pointers on the stack:

Device number (P1)

Command (P2)

Data (P3)

Result (P4)

Call:

Call the Control Device via the "Long Jump" in the Pointer area. Absolute address 00600 hex.

Exit:

Data or Status (P4) or at (P3:P4)

The interface for the Control Device expects the stack to contain 4 word pointers to each of the entry parameters given above. The order in which the parameters are to be pushed onto the stack is Device, Command, Data and finally Result.

Normally, as in BASIC for example, the language processor will be responsible for setting up the stack with the relevant pointers. The applications program must simply ensure that the parameters are supplied in correct order and number.

The DX:SI Data pointer facility is also available in High level access by combining the two pointers P3:P4 to give Segment:Offset respectively or Double word data.

It is of particular importance to note that pointers within most language processors, and once again BASIC is a good example, are relative to the processor's own Data Segment. There is usually no direct method for the Application to obtain the value of the Segment. In such cases the Application must resort to machine code subroutines to ensure a correct interface. Refer to the Appendix on Language Interfaces for details of how to do this and further information regarding interfaces.

Apart from these differences, the Control Device is accessed and reacts in the same way whether at Low or High level.

High level Control device access

Accessing the Control Device from BASIC

The following sections provide examples of how to use the Control Device from BASIC. Here we can study the general concept of implementing I/O functions from BASIC.

```
 10 REM – Concept of programming Control Device
 20 DEF SEG=&H60  'Set current segment to 60 hex
 30 IO=0          'Pointer to offset 0 within segment
 40 DEV%=1        'Device number 1
 50 COM%=0        'Command is 0 i.e. initialise
 60 DAT%=0        'Data
 70 RET%=0        'Data/Status return
100 CALL IO(DEV%,COM%,DAT%,RET%)
```

The CALL will vector to absolute address 00600 hex.

The stack will contain the following:

Word pointers:	Device number	(DEV%) i.e.	P1
	Command	(COM%)	P2
	Data	(DAT%)	P3
	Return data	(RET%)	P4

"pushed" in the order given followed by a "Double word" return address.

The Control Device will validate the parameters and route the I/O request to the specified routine.

Execution of the command will normally result in Data/Status being returned to the application program via the RET% variable.

```
200 SEG%=&HC000  'Segment pointer to a buffer
210 OFF%=0       'Offset within buffer
220 DEV%=10:COM%=3
250 CALL IO(DEV%,COM%,SEG%,OFF%)
```

The example follows on to show the alternative entry parameters where the SEG% (P3) & OFF% (P4) variables represent a double word Segment:Offset pointer to the data.

Any return data or status is returned relative to the data pointer.

Note: The references to device numbers, commands etc, given in the examples are purely arbitrary. The exact specification for each individual driver is described below.

Errors

On exit from the Control Device, PX will be set to a value to indicate the status or data returned from the individual command. Where data is returned, details are given with each call.

Where a status is returned then it will be one of the following unless otherwise stated:

0000 - Device present and on-line
8000 - Write protect
8001 - Unknown unit
8002 - Not ready
8003 - Unknown command
8004 - CRC error
8005 - Bad request length
8006 - Seek error
8007 - Unknown media
8008 - Sector not found
8009 - Printer out of paper
800A - Write fault
800B - Read fault
800C - General failure
FFFF - Device error/invalid device or command

In certain cases the 0200 hex bit is set to indicate that the device is busy.

Specific application

This section details each individual hardware driver and how it is accessible via the Control Device.

The section is limited to the definition of each call. A full description of the driver and examples of how to use its functions from a high level language (and implicitly a low level language) are given in the following chapters on the Drivers.

The Device number is given with the heading of each driver and not included in the definition to avoid repetition. It must be stressed however that this parameter *MUST* be supplied with every call to the Control device.

For ease of identification and to provide a common approach to each method of invoking the Control Device, each parameter will be given an identifier as follows:

P1 - Device number

P2 - Command

P3 - Data or Data Pointer Segment

P4 - Data or Data Pointer Offset

PX - Data or Status

PTR - Pointer (Double word Segment:Offset)

In addition to the parameters above the following identifiers have a common meaning in the driver sections:

SEG - Segment
OFF - Offset
SET - Current setting.

A number of commands are of the format 'Get/Set an attribute' and successful execution results in the PX parameter being returned SET (i.e. the current setting). If P3 is invalid then the setting, which is returned in PX, remains unchanged.

Note: Invalid P3 settings may be conveniently used to obtain the current setting.

Certain calls in the Generic Control Device are not available on all machines within the family. Availability is denoted in the M/C (machine column):

A is available on Apricot or Xi
P is available on Portable
F is available on F1

Device numbers

The Device numbers are:

ASCII	Hex	Device
1	31	Screen
2	32	Keyboard
3	33	Microscreen (only on Apricot PC/Xi)
4	34	Serial I/O
5	35	Parallel I/O
6	36	Mouse
7	37	Clock
8	38	Sound
9	39	Floppy Disk
@	40	Winchester
A	41	Modem (not implemented in Control Device)
B	42	Cache/Graphics/IBM (only on Apricot PC/Xi)

Screen - P1 = 31 hex

M/C	P2	Function	P3	P4	PX	Comment
PFA	0000	Initialise driver			0000 OK 0001 No driver	
A	0001	Get/Set Text mode Get/Set Graphics mode	0000 0001		SET SET	note 1
A	0002	Get/Set display on Get/Set display off	0000 0001		SET SET	note 2
	0003	No-op				
PF	0004	Print character 'cc'	00cc			note 3
PF	0005	Print string	SEG	OFF		note 4
PF	0006	Reserved				
PF	0007	Reserved				
PF	0008	Reserved				
PF	0009	Reserved				
PF	000A	Reserved				
PF	000B	Reserved				
PF	000C	Reserved				
PF	000D	Reserved				
PF	000E	Output char. to Default screen.	rrcc	FORMX		note 5
PF	000F	Update Screen Bit Image from Default character Image				note 5/6

Key: rrcc rr = screen row
cc = screen column

Screen driver notes

1. Only available on pc/Xi where text mode is a feature.
2. Switch Display OFF/ON. Suitable for pseudo rapid Display changes.
3. The use of software Interrupt F1 hex is faster. This is a logical print, i.e. characters < 20 hex are obeyed.
4. SEG:OFF is a pointer to a 3 word table:

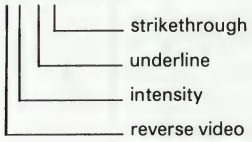
   Word 0 - Number of characters
   Word 1 - Offset of string
   Word 2 - Segment of string

5. FORMX - A word to represent the current Default Screen. Three formats are possible:

   Apricot Compatible 80 column:

   Lo byte - 8 bit ASCII data

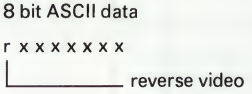
   Hi byte - r i u s x x x x

   Apricot 40 column:

   Lo byte : 8 bit ASCII data

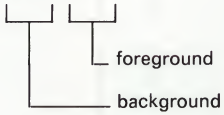
   Hi byte : r x x x x x x x

   Colour:

   Lo byte : 8 bit ASCII data

   Hi byte : x b b b x f f f

   Refer to Screen Driver chapter for a definition of Apricot Compatible mode.

6. This is a rapid update of the Screen Bit Image from the Default Screen Character Image. It is used by the Screen Driver following such functions as scrolling.

Keyboard - P1 = 32 hex

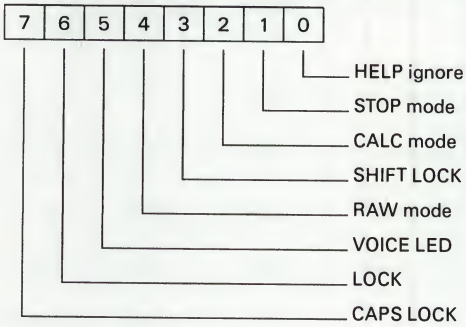
M/C	P2	Function	P3	P4	PX	Comment
PFA	0000	Initialise			0000 0001	OK no driver
PFA	0001	Disable auto rpt Enable auto rpt	0000 0001		SET SET	
A	0002	Set auto rpt lead-in delay to xx 20ms intervals	00xx		SET	
A	0003	Set auto rpt rate to xx 20 ms	00xx		SET	
PFA	0004	Set Fall-through mode : OFF ON	0000 0001		SET SET	note 1 note 2
PFA	0005	Flush queue			0000	note 3
PFA	0006	Reset CTRL/SHIFT status & clear down-code buffer			0000	note 4
PFA	0007	Get/Set HELP ignore mode: OFF ON	0000 0001		SET SET	
PFA	0008	Place data xx in queue		00xx	0000 FFFF	OK Full/BELL
PFA	0009	Sound BELL			0000	
PFA	000A	Return Queue byte count			00xx	xx = number
PF	000B	Get byte from Queue Wait if queue empty			00xx	xx = byte
PF	000C	Look-ahead. Get Next byte.			00xx FFFF	xx = byte empty
PF	000D	Add string to queue	SEG	OFF	0000 FFFF	OK - note 5 Full/BELL
PF	000E	Get string from queue	SEG	OFF	0000	note 6
PF	000F	Get/Set Status	aaoo		00xx	note 7

Keyboard notes

1. Down-codes are translated and then put in the queue.
2. Down-codes are not translated. They are placed in the queue "raw".
3. Remove all data from queue.
4. Set CTRL and SHIFT to not affective and clear down codes in queue. Use after Keyboard Table change.
5. SEG:OFF is a pointer to a 3 word packet consisting:

   WORD = No of bytes in string (max 80 - no check)
   DWORD = Seg:Off pointer to string

   If there is not enough room for the complete string in the buffer then it is rejected, the bell is sounded and the status is set to FFFF hex.
6. As 5 except for remarks on overflow. The byte count in this case must be pre-specified and the call will not return until the specified Key count.
7. The status of the keyboard is configured in a byte as follows:



where 0 = off 1 = on

The high byte of P3 (aa) is logically AND'ed with the status, i.e. gets the current status, whereas the low byte (oo) is logically OR'ed with the status and hence sets it.

e.g P3 = FF00H will return status only
P3 = FD02H will set STOP on

Serial I/O - P1 = 34 hex

M/C	P2	Function	P3	P4	PX	Comment
PFA	0000	Initialise driver			0000	
PFA	0001	Xmit character xx	00xx		0000 FFFF	OK Fail
PFA	0002	Rcve character			FFFF 00xx	char avail. OK
PFA	0003	Update SIO settings			0000	note 1
PFA	0004	Get/Set Xmit baud rate (x = 1 - F)	000x		SET	note 2
PFA	0005	Get/Set Rcve baud rate (x = 1 - F)	000x		SET	note 2
PFA	0006	Get/Set Xmit bits/char. (x = 5 - 8)	000x		SET	note 3
PFA	0007	Get/Set Rcve bits/char. (x = 5 - 8)	000x		SET	note 3
PFA	0008	Get/Set Stop bits/char. (x = 1 - 3)	000x		SET	note 4
PFA	0009	Get/Set Parity type. (x = 0 - 2)	000x		SET	note 5
PFA	000A	Get/Set Xmit XON/XOFF	000x		SET	0 = disable 1 = enable
PFA	000B	Get/Set Rcve XON/XOFF	000x		SET	0 = disable 1 = enable
PFA	000C	Get/Set RTS (Ready to send)	000x		SET	0 = reset 1 = set
PFA	000D	Get/Set DTR (Data Term ready)	000x		SET	0 = reset 1 = set
PFA	000E	Get/Set Xmit enable	000x		SET	0 = disable 1 = enable
PFA	000F	Get/Set Rcve enable	000x		SET	0 = disable 1 = enable note 6
PFA	0010	Get CTS (Clear to send) status			0000 0001	Reset Set
PFA	0011	Get DCD (Data Carr. Detect)			0000 0001	Reset Set
PFA	0012	Get DSR (Data Set Ready) status			0000 0001	Reset Set
PFA	0013	Get/Set xx nulls after CR	00xx		SET	note 7
PFA	0014	Get/Set xx nulls after FF	00xx		SET	note 7
PFA	0015	Get/Set auto LF after CR	000x		SET	0 = disable 1 = enable

Serial I/O notes

1. All calls to configure the SIO are not activated until this command is executed. Specifically these calls are 0004 through 0009, 0018 and 0019.
2. P3 is set to a value (x) in the range 1 - F hex corresponding to an entry in the table below:

Value	Baud Rate
1	50
2	75
3	110
4	134.5
5	150
6	300
7	600
8	1200
9	1800
A	2400
B	3600
C	4800
D	7200
E	9600
F	19200

3. P3 is set to a value (x) in the range 5 - 8 to denote the number of data bits per character.
4. P3 is set to a value (x) in the range 1 - 3 corresponding to an entry in the table below:

1 - 1 stop bit
2 - 1.5 stop bits
3 - 2 stop bits

5. P3 is set to a value (x) in the range 0 - 2 corresponding to a parity type as given in the table below:

0 - none
1 - odd
2 - even

6. These calls directly enable/disable the SIO. Care must be taken to ensure that the SIO has completed all outstanding communications. This can only be acieved by either monitoring the Read Registers directly and looking at the "ALL SENT" flag or by use of Call 24 hex which does not return 0 until "ALL SENT".
7. P3 is set to a value (xx) in the range 0 to FF hex. This command is of use in conjunction with transmission to printer devices. It generates a series of null characters (10 times the xx number specified) following a CR or FF as designated.

Serial I/O - P1 = 34 hex (continued)

M/C	P2	Function	P3	P4	PX	Comment
PFA	0016	Enable serial mouse			0000	note 8
PFA	0017	Disable serial Mouse			0000	note 8
PFA	0018	Get/Set RTS/CTS protocol	000x		SET	0 = disable 1 = enable 2 = auto-enable note 9
PFA	0019	Get/Set DTR/DSR protocol	000x		SET	note 10
PA	001A	Get/Set External SIO control	000x		SET	note 11
PF	001B	Rcve Queue Look-ahead			FFFF 00xx	none Rcve char.
PF	001C	Flush Xmit Queue			0000	note 12
PF	001D	Flush Rcve Queue			0000	note 12
PF	001E	Get Rcve Char. Status-reset error.F			00xx	note 13
PF	001F	Xmit string	SEG	OFF	0000 xxxx	OK Error code Note 14
PF	0020	Rcve string	SEG	OFF	0000	Note 14
PF	0021	Get/Set Rcve Queue length	xxxx		SET	Note 15
PF	0022	Get/Set Xmit Queue length	xxxx		SET	Note 15
PF	0023	Return count chars in RCVE Queue			xxxx	
PF	0024	Return count chars in XMIT Queue			xxxx	

Serial I/O notes (continued)

8. When the serial mouse is enabled the SIO interrupt handler vectors all RX data to the Serial Mouse Interrupt (0FA hex) handler. Control is returned to the RS-232 when the serial mouse is disabled.

9. RTS/CTS and DTR/DSR protocols are supported by the SIO directly when Auto-enables mode is invoked. In Auto-enables mode the CTS line serves a specific function, i.e. no transmission takes place until CTS goes low. If CTS is used for any other purpose then Auto-enable should not be invoked.

10. Refer to note 9. In addition both enable/disable in this call de-activates Auto-enable in both protocols.

11. This command allows the application to switch the SIO between BIOS control and user control. The SIO under BIOS, for example, is only in ASYNC - in order to program the SIO in SYNC mode the user must take control of the SIO and re-program it. The entry parameter (P3) selects one of the following:

    0 - SIO under BIOS control
    1 - SIO under interrupt control (F0 hex)

    In the latter case the interrupt routine F0 hex must handle all Tx, Rx and errors related to the SIO.

    The return cell (PX) gives details of the SIO:

    PX low byte - Base SIO port address

    The offsets from the base are:

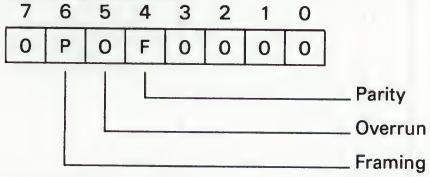
    [base + 0] = Ch A data
    [base + 2] = Ch A status
    [base + 4] = Ch B data
    [base + 6] = Ch B status

    PX high byte - has one of 2 values i.e.

    0 = RS-232 on Ch A
    1 = RS-232 on Ch B

12. These calls should be invoked if spurious data is transmited/received.

13. Returns status in PX low byte as per SIO Read Register 1 (refer to Hardware section).
    This is a 'running' error status on all RX characters since the last status call. The call clears the status.



    The status is used to detect errors in RX Blocks of characters.

    Note that all other bits are clear. The call cannot be used to monitor 'ALL SENT'.

14. Parameters P3 & P4 form an address pointer SEG:OFF to a 3 word packet of the format:

    WORD - number of bytes in string
    DWORD - pointer to string (SEG:OFF)

15. Set the Queue lengths to a value xxxx in the range 1 - 512 bytes.

Parallel I/O - P1 = 35 hex

M/C	P2	Function	P3	P4	PX	Comment
PFA	0000	Initialise			0000	
PFA	0001	Return available buffer space.			xxxx	note 1
PA	0002	Get/Set FAULT line status detect.	000x		SET	0 = disable 1 = enable note 2
PA	0003	Get/Set SELECT line status detect.	000x		SET	0 = disable 1 = enable
PA	0004	Get/Set PAPER OUT status detect.	000x		SET	0 = disable 1 = enable
PFA	0005	Get/Set Auto LF after CR enable	000x		SET	0 = disable 1 = enable
PFA	0006	Get/Set Serial/Parallel Device	000x		SET	0 = Parallel 1 = Serial note 3
PFA	0007	Print character on output device.	00cc		xxxx	cc = character PX = status note 4
PF	0008	Flush printer output buffer			0000	note 5
PF	0009	Transmit string	SEG	OFF	xxxx	PX = status note 6
PF	000A	Return output status			0000 xxxx	OK Printer busy Paper out

Parallel I/O notes

1. The buffer has a capacity of 2k bytes. If the value xxxx in PX is less than 3 then regard as full.
2. Detection of the 3 error lines:

   FAULT
   SELECT
   PAPER ERROR

   may be individually enabled or disabled.
3. By default the BIOS is vectored to the parallel port. This toggle switch allows re-direction to the Serial port. In doing so the parallel printer buffer (2 Kbytes) is then used for Serial Output.

   Serial output is mixed with parallel output.
4. If the character is received by the current output support driver and successfully output the PX is returned zero. Otherwise PX contains an error code.
5. Resets pointers to beginning of buffer. Ensure that printing has been completed under normal circumstances by querying the number of free bytes in the buffer.
6. P3:P4 form a pointer to a 3 word packet as follows:

   WORD - byte count
   DWORD - string pointer
7. If the printer is busy or Paper out detected then PX will be set to an error code.

Mouse - P1 = 36 hex

The Mouse is not directly supported by the Control Device. It is a loadable Device driver.

However calls to the Control Device are provided to enable and disable the Serial Mouse and these can be found in the section Serial I/O.

Clock - P1 = 37 hex

M/C	P2	Function	P3	P4	PX	Comment
PF	0000	Initialise-reset time to 0:00:00 1-1-80			0000	
PF	0001	Set Date/Time	SEG	OFF	0000	
		P3:P4 points to data block: WORD - Days since 1-1-80 BYTE - Minutes BYTE - Hours BYTE - Hundredths of secs. BYTE - Seconds				
PF	0002	Get Date/Time	SEG	OFF	0000	
		P3:P4 points to data block as command 2.				

Sound - P1=38 hex

M/C	P2	Function	P3	P4	PX	Comment
PFA	0000	Initialise			0000	
PFA	0001	Get/Set Key click volume	000x		SET	note 1
PFA	0002	Get/Set Bell volume	000x		SET	note 1
PA	0003	Get/Set Channel 2 volume	000x		SET	note 1
PA	0004	Get/Set Channel 3 volume	000x		SET	note 1
PA	0005	Get/Set Channel 2 frequency	0xxx		SET	note 2
PA	0006	Get/Set Channel 3 frequency	0xxx		SET	note 2
PFA	0007	Get/Set Bell frequency	0xxx		SET	note 2
PFA	0008	Get/Set Bell duration	00xx		SET	note 3
PF	0009	Sound Bell			0000	
PF	000A	Sound Click			0000	

Sound notes

1. The value of x determines the volume within a scale of 0 to 15 where 0 = maximum and 15 = minimum (figures in decimal).
2. The value of xxx determines the frequency within a scale of 0 to 1023 decimal where 0 = maximum and 1023 is minimum.
3. The value of xx determines the duration of the Bell sound in multiples of 20 ms. The value of xx is in the range 0 to 255 decimal.

Floppy disk - P1 = 39 hex

M/C	P2	Function	P3	P4	PX	Comment
PFA	0000	Initialise driver			xxxx	note 1
PFA	0001	Set Drive number	000x		FFFF SET	Invalid drive 0 = drive 0 1 = drive 1
PFA	0002	Set segment address of Track Image for format	SEG		0000	note 2
PFA	0003	Set offset address of Track Image for format	OFF		0000	note 2
PFA	0004	Format floppy disk x (0 = 70ss/1 = 80ss/2 = 80ds)	000x		0000 xxxx	OK Error status note 1
PFA	0005	Return status of disk x	000x		xxxx	note 3
PFA	0006	Set disk x status to swapped	000x		0000 FFFF	OK invalid drive
PFA	0007	Return disk type in drive x	000x		0000 0001 0002	70 track SS 80 track SS 80 track DS notes 1/5
PF	0008	Return drive type of drive x	000x		0000 0001 0002	70 track SS 80 track SS 80 track DS note 1/5
PF	0009	Check if disk swapped in drive x	000x		0000 0001 0002	not swapped swapped no disk note 4
PF	000A	Get BPB (do this whenever disk is swapped)	SEG	OFF	xxxx	PX = Status note 6
PF	000B	Read/Write/Verify/ Write + Verify	SEG	OFF	xxxx	PX = Status note 7
PF	000C	Return status of drive x	000x		xxxx FFFF	Status Invalid note 8

Floppy disk notes

1. Status - Refer to the Errors section above for a list of errors and status settings.
2. Track image is controller dependant. Refer to Hardware section.
3. Apricot Compatible version will - not be supported in future releases. Returns the status register of the device. Refer to the hardware section.
4. This command clears the "swapped" flag! *IMPORTANT - refer to the Disk Driver chapter Applications Interest section.*
5. If the unit is not identifiable then the status returned is 8001 hex.
6. Whenever a disk has been swapped then this call should be executed. The P3:P4 parameters sepcify a Segment:Offset pointer to a 3 word packet:

   WORD - Drive number (0 or 1)
   DWORD - Pointer to a 512 byte scratch area

   The status PX will be zero if the command was executed correctly otherwise refer to the section on Errors above for detail. *IMPORTANT - refer to the Disk Driver chapter Applications Interest section.*
7. The P3:P4 parameters form a pointer to a block of 6 words which has the following format:

   WORD - Drive number (0 or 1)
   WORD - Command
   0 = READ
   1 = WRITE
   2 = VERIFY
   3 = WRITE with VERIFY
   WORD - Address of first logical sector
   WORD - Number of sectors to transfer
   DWORD - Buffer address (not needed for Verify)

   The number of sectors field is set to the number of sectors actually transferred.

   The status PX will be zero for a successful operation otherwise refer to section 2.4 Errors for detail.
8. THIS CALL SUCCEEDS CALL 0005. It should be used in preference to it! Status is as for CALL 0005 - refer to Hardware section.

Winchester - P1 = 40 hex

M/C	P2	Function	P3	P4	PX	Comment
PF	0000	Initialise drivers			xxxx	Note 1
PF	0001	Set Disk x status to swapped	000x		0000 FFFF	OK Invalid drive note 2
PF	0002	Return drive type of drive x	000x		0000 0001 0002	5 megabyte 10 megabyte 20 megabyte note 3
PF	0003	Check if drive x disk swapped	000x		0000 0001 0002	Not swapped Disc swapped No disk note 2
PF	0004	Get BPB (do this whenever disk is swapped)	SEG	OFF	xxxx	PX = Status note 4
PF	0005	Read/Write/Verify/Write+Verify	SEG	OFF	xxxx	PX = Status note 5

Winchester notes

1. Status - Refer to the section on *Errors* for a list of errors and status settings.
2. The "swapped" status on Winchester is used to indicate a change in state, e.g. the Winchester has been formatted.
3. If the unit is not identifiable then the status returned is 8001 hex.
4. Whenever a disk has been swapped then this call should be executed. The P3:P4 parameters sepcify a Segment:Offset pointer to a 3 word packet:

   WORD - Drive number (0 or 1)
   DWORD - Pointer to a 512 byte scratch area

   The status PX will be zero if the command was executed correctly otherwise refer to section on Errors for details. IMPORTANT - refer to the Disk Driver chapter section Applications Interest.
5. The P3:P4 parameters form a pointer to an a block of 6 words which has the following format:

   WORD - Drive number (0 or 1)
   WORD - Command
   0 = READ
   1 = WRITE
   2 = VERIFY
   3 = WRITE with VERIFY
   WORD - Address of first logical sector
   WORD - Number of sectors to transfer
   DWORD - Buffer address (not needed for Verify)

   The number of sectors field is set to the number of sectors actually transferred.

   The status PX will be zero for a successful operation otherwise refer to section on Errors for detail.

Screen driver 3.3

Contents

Overview

The Apricot Portable features a number of Display options. The 80 column by 25 row LCD display is an integral part of the hardware. An optional Colour display can also be connected. Both displays are supported by the Screen Driver but not simultaneously.

The minimum hardware configuration provides for:

1. Apricot Compatibility on the LCD, or
2. 4 Colour on the Display

Optional hardware is available for the Apricot to provide for up to 16 Colours.

The Screen driver limits the colour selection on the Apricot to a maximum of 8 from the 16 available colours in order to provide for the LCD.

The Hardware section of this manual provides a complete description of the Screen RAM. Applications wishing to implement features outside the support of the BIOS, such as 16 colour, should reference this section.

The Screen driver provides a Generic interface for applications through the Control Device and the use of ESCape sequences.

Apricot compatibility, i.e. the character attributes of Reverse, Strikethrough, Underline and Intensity, is provided in monochrome only.

The Screen environment, apart from the features mentioned above, also provides a 256/200 scan line mode to support countries with 50/60 Hz electrical standards. The Screen driver supports this feature by using different Font sizes.

Other features available in the Screen Driver provide the Application with complete control of the LCD and Colour Display. The principal ones are tabled below:

A comprehensive set of ESCape sequences including a subset of the ANSI standards.

A 16 Colour programmable palette, which depending upon hardware configuration, allows the Driver to support the following:

1. Monochrome (2 colours from 16)
2. Colour (4 colours from 16)
3. Colour (8 colours from 16)

Foreground, Background character and Screen base Colour selection.

Partial Windows.

Hard copy.

WP primitives e.g. L & R scrolling, Multi character insertion, etc.

Switchable character Fonts. Two Fonts are provided as standard to cater for the line modes outlined above. Applications may use their own character Fonts by loading them and designating them active.

Finally, the Apricot does not support hardware text. Instead it uses a Bit orientated Screen RAM. For this reason the Screen driver uses a number of character Images held within the memory as described in the Guide to the BIOS chapter.

The following sections detail the above features and provide examples to assist in usage.

Applications interest

Screen images

The LCD and the Display each have an area reserved in memory called an "Image" screen. The Pointer table has an entry to point to each one.

The image consists of a "word" for every location on the screen. Each one contains the Ascii value of the character together with its attributes.

The attributes vary according to whether the Screen Driver is in Apricot compatible monochrome mode or in colour. Later sections detail these attributes.

The image is used mainly to support hardcopy screen dumps, "return of the character at the current cursor position" and left/right scrolling.

The real screen RAM is in another location and contains a bit image for each character. In order to distinguish between the two images the real screen RAM is termed the "Bit Image" whereas the character image is termed the "Screen Image".

The Bit Image is generated from the active Font table. This is discussed further under Systems interest.

When requests are made to output data to the screen then both the Screen Image and the Bit Image are updated by the Screen driver.

The Control Device provides calls which enable the Screen driver and Applications to compose a screen together with its attributes and then use a single command to update the Bit Image from the Screen Image.

Using ESCape sequences

The ESCape sequences are detailed in two sections within the manual. Appendix E lists them in quick reference form broken down by activity.

They are also detailed in a later section as a Table in ascending Ascii order.

The ESCape sequences provide the main control features on the screen such as:

1. Screen characteristics e.g. intensity, underline ...
2. Cursor control e.g. position, UP, DOWN, LEFT ...
3. Colour e.g. set palette, foreground/background ...
4. WP primitives e.g. Insert line, Delete line ...
5. Hard copy
6. ANSI, a subset of the standard sequences
7. Fast and slow screen scrolling

The ESCape sequences form a Generic interface across the Apricot family. For Applications writers wishing to produce compatible software, each ESCape is annotated with one or more of the following keys to indicate availibility on a particular machine:

P - Portable
F - F1
A - Apricot PC & Xi

Examples of how to use ESCape sequences are to be found throughout the manual. For readers unfamiliar with the principles, a full explanation can be found at the beginning of the section ESCape Sequence Table. To assist in the reading of the following sections a short example follows:

Example: How to use an ESCape sequence

ESCape sequences consist of a series of pre-defined characters which are sent to the Screen driver by the Application, typically as in Basic by the conventional PRINT statement. The sequence is prefixed with a special character, the ESCape character, whose decimal value is 27. It is a non-printable character.

The next character in the sequence is the command. This in turn may be followed by one or more characters depending upon the individual command.

Usually these sequences make programs difficult to read and tend to clutter examples. For this reason examples in this manual define the command sequence in a string which is given a meaningful name.

Each string may then be used freely with or without parameters as individual commands dictate.

```
10 CLS$=CHR$(27)+"E"      'Clear screen command (no parameters)
20 ENV$=CHR$(27)+"7"      'Set environment (needs 1 parameter)
100 PRINT CLS$            'Clear the Screen
110 PRINT ENV$+"4";"Monochrome Display"
```

Statements 10 and 20 pre-define the commands for the reason stated above. Statement 100 executes the Clear screen ESCape command and then statement 110 invokes the 80 column monochrome Display environment. Note that this requires an additional parameter of "4".

The full meaning of the Screen environment is discussed in the next section and uses the definitions formed above.

For readers not familiar with using ESCapes the long hand way of writing statement 110 is given below.

```
110 PRINT CHR$(27)+"7"+"4";"Monochrome Display"
```

In the short term there would seem to be little advantage in using the methods given above. However, more complex examples later in the manual demonstrate the benefits in terms of readability.

Screen environment

The screen environment may be modified at run time by using an ESCape sequence. There are 3 modes which may be selected to enable the Application to switch Screens, and select functions such as; the number of colours, the width of the Screen and Apricot compatibility mode. They are:

ENV$+"0" 80 column LCD Apricot monochrome compatability.

The Cursor is Homed and any image already on the LCD remains unchanged. All output following is then directed to the LCD.

If switching from a Display environment then the Display's Cursor is turned off. The Display image will remain unchanged.

ENV$+"1" 80 column 4 or 8 colour Display

The LCD Cursor is turned off. Any image on the LCD will remain.

All Screen output is directed to the Display in 80 column mode in colour as required. Apricot Compatibility is not supported.

ENV$+"2" Reserved

ENV$+"3" 80 column Display Apricot monochrome compatability

The LCD Cursor is switched off.
Any image on the LCD remains unchanged.

The Display is cleared and the Cursor is Homed.

All output in this mode is directed to the Display. The features of Apricot compatibilty are available and in addition two colours are supported for background and foreground.

Reference to the Keyboard driver chapter shows that these sequences have been implemented in the default version of the keyboard table, i.e. the environment may be set by pressing certain keys.

The examples in Basic later in the section illustrate the use of these ESCape sequences.

The display may be driven in either 200 or 256 scan line mode whereas the LCD is fixed at 200 lines only.

The line mode is determined at BOOT time by a Configuration constant and cannot be modified during run time.

These line modes are supported by the incorporation of two character Font tables within the user RAM.

The Font options are:

1. 8 x 8 for the 200 line
2. 8 x 10 for the 256 line

The BOOT routines select the Font according to the constant in the configuration table and modify the pointers to reflect the "active" Font. The screen is driven by default in the 80 column LCD Apricot compatible mode.

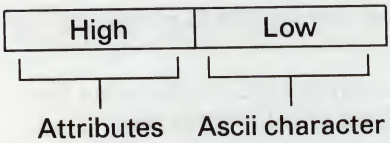
Apricot compatible mode

The Apricot compatible mode supports the following features:

1. Monochrome
2. Character attributes:
   —Reverse video
   —Intensity
   —Underline
   —Strikethrough
3. Word Processing primitives.
4. Screen dump
5. Partial windows

Apricot compatible mode is not available when the Screen environment is set to full colour, i.e. either 4 or 8 colours from the 16 colours available.

The Screen Image has one word for each character on the screen which holds the Ascii value of the character together with its attributes as illustrated below:



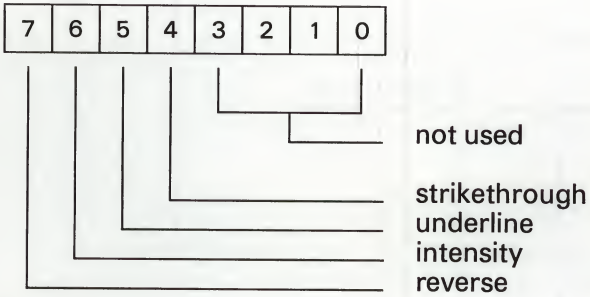
The attributes are used to define either the character feature stated above or the definition of background/foreground colour but not both together.

In 80 column Apricot compatible mode, however, two colours are supported as well as the character features. The attributes are not used to define the colours in this case but are reserved for the Apricot features.

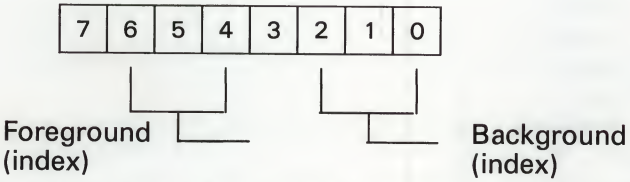
The possible attribute uses are illustrated below:

The attributes have the following settings:

APRICOT compatible -



Colour -



The index is a reference to the colour palette. This is discussed in the next section.

Colour

The colour palette provides a selection of 16 colours. Each colour is designated a Colour code which is an Ascii value as given below:

Colour	Ascii Colour code
Black	0
Blue	1
Green	2
Cyan	3
Red	4
Magenta	5
Brown	6
Light Grey	7
Dark Grey	8
Light blue	9
Light green	:
Light cyan	;
Light red	<
Light magenta	=
Yellow	>
White	?

Each entry in the palette is referenced by an index which is also an Ascii value. The palette is assigned default values on Screen driver initialisation. The following table depicts the default settings for each index.

Ascii index	Ascii Colour code	Colour
0	9	Light Blue
1	?	White
2	0	Black
3	<	Light Red
4	:	Light Green
5	>	Yellow
6	;	Light Cyan
7	=	Light Magenta
8	1	Blue
9	7	Light Grey
:	8	Grey
;	4	Red
<	2	Green
=	6	Brown
>	3	Cyan
?	5	Magenta

The hardware configuration and screen environment dictate how many colours may be used, i.e.

2 for Apricot Compatible monochrome

4 or 8 for Colour

The significant colours in each of these are taken to be in the first x entries of the palette where x = 2, 4 or 8.

Note:

1. In all environments the first entry (index "0") is the basic background colour referred to later as the "screen colour".
2. In the 2 colour Apricot compatible mode the following two entries have a significance:

   index"0" - Background
   index"1" - Foreground
3. The above default settings provide an acceptable contrast for all Applications regardless of the Display attached.

The setting of the palette is achieved via ESCape sequences, as the following example shows:

Example: Apricot compatible and full colour

```
10 REM - setting the Colour palette
20 CLS$=CHR$(27)+"z"                    ' clear screen
30 ENV$=CHR$(27)+"7"                    ' environment ESC
40 RES$=CHR$(27)+"xG"                   ' reset palette
50 COL$=CHR$(27)+"]"                    ' set palette ESC
70 FORE$=CHR$(27)+"5"                   ' foreground ESC
80 BACK$=CHR$(27)+"6"                   ' background

100 PRINT CLS$;RES$;ENV$;"3"            ' Apricot compatible 2 colour.

110 PRINT COL$;"01";                    ' index "0" blue
120 PRINT COL$;"1>";                    ' index "1" yellow
130 PRINT "Yellow on Blue background"

140 INPUT "Press return key"; A         ' just to pause

200 PRINT CLS$;RES$;ENV$;"1"            ' 4 or 8 colour
                                        ' Port.- 8 colour

300 PRINT COL$;"02";                    'screen base green
310 PRINT COL$;"14";                    'index 1 red
320 PRINT COL$;"29";                    'index 2 light blue
330 PRINT COL$;"3>";                    'index 3 yellow

400 PRINT BACK$;"1";                    'char. background red
410 PRINT FORE$;"3";                    'char. foreground yellow

500 PRINT "Yellow characters on Red background"
510 INPUT "Press return"; A             ' just a pause

600 PRINT BACK$;"3"                     'char. background yellow
610 PRINT FORE$;"2"                     'char. fore. light blue

700 PRINT "Light Blue chars. on yellow background"
```

Notes:

1. Statement 100 sets the environment to 2 colour Apricot compatible mode. The background and foreground colours are defined in statements 110 and 120 respectively.

2. Statement 200 switches the environment to full colour. Although 8 colour is possible, the example is designed to run on the minimum hardware configuration and is confined to 4 colours.

To illustrate the above programming example further the palette is shown below in its re-programmed form for the two different settings generated in the example.

Palette after statement 110 & 120:

Ascii index	Ascii Col. code	
0	1	Blue
1	>	Yellow
2	0	unchanged
thru		
?	5	unchanged

Palette after statement 330 is:

Ascii index	Ascii Col. code	
0	2	Green
1	4	Red
2	9	Light Blue
3	>	Yellow
4	:	unchanged
thru		
?	5	unchanged

Note: Index 0 is the Screen base colour.

ANSI ESCape sequences

A subset of the ANSI escape sequences is supported by the generic drivers through the use of ESCape sequences. The ESCape sequence is

CHR$(27)+"["+ parameters

The functions available are:

Mnemonic	Param.	Code	Function
CU			Cursor movement
CUU	n	A	n lines UP
CUD	n	B	n lines DOWN
CUF	n	C	n columns forward
CUB	n	D	n columns back
CUP	y;x	H	position cursor at y;x
HVP	y;x	f	position cursor at y;x (IBM compatible)
ER			Text erase:
ED	0	J	Erase-cursor to end of screen
ED	1	J	Erase-start screen to cursor
ED	2	J	Clear screen. Cursor not moved.
EL	0	K	Erase-cursor to end of line
EL	1	K	Erase-cursor to start of line
EL	2	K	Erase-entire line of cursor
CP			Report functions:
CPR	1;c	R	Cursor position to keyboard buffer
SM			Set mode:
LNM	20	h	Auto CR (on receipt of LF)
DECOM	6	h	Origin mode on
DECAWM	7	h	Auto-wrap at end of line
RM			Reset mode:
LNM	20	l	No auto CR
DECOM	6	l	Origin mode off
DECAWM	7	l	No auto-wrap
TAC			Text attributes
SGR	0	m	All attributes off
SGR	1	m	Bold ON
SGR	4	m	Underline ON
SGR	7	m	Reverse ON
SCR			Screen size/mode
DSR	6	n	Device status report
DECSTBM	T;B	r	Set T (top) and B (bottom) lines
SCP		s	Save cursor position
RCP		u	Restore cursor position

The ANSI ESCape sequence parameters are required in the order:

CHR$(27) + "[" + 'param list' + 'Code'

Where there is more than one parameter a list is required with each parameter separated by a ";" (semi-colon).

Example: Cursor movement

```
10 AN$=CHR$(27)+"["    ' ANSI lead-in

20 PRINT AN$;"2J"  ' clear the screen
30 PRINT AN$;"10;15H";

40 PRINT "print from line 10 column 15"

50 PRINT AN$;"5A";
60 PRINT "now from line 5 column 1"

70 PRINT AN$;"10;14r";
80 PRINT "screen 5 lines high i.e lines 10-14"
```

Windows and Cursor addressing

A window may be designated in any rectangular block on the screen. The remainder of the screen remains unchanged.

All Print, scrolling and WP primitives will restrict themselves to the bounds of the window.

Direct cursor positioning is relative to the origin of the physical screen, not the window. However, the driver will not allow the cursor to be moved outside the window.

Example: Outline a window

```
  1 width=255
 10 esc$=chr$(27)                          ' ESC
 20 cls$=esc$+"E"                          ' clear window or screen
 30 window$=esc$+","                       ' to set window
 40 nowind$=esc$+"."                       ' cancel window
 50 cur$=esc$+"Y"                          ' position cursor
100 print cls$;nowind$;                    ' clear screen - no window
110 input "top line                        : "; top
120 input "bottom line                     : "; bot
130 input "left margin                     : "; lft
140 input "right margin                    : "; rgt
                                           ' do not allow 0 coordinate
150 print window$+chr$(31+top)+chr$(31+bot)
                +chr$(31+lft)+chr$(31+rgt);
200 print esc$+"p";                        ' reverse video to emphasise
300 for row=top to bot
310 for col=lft to rgt
320 print cur$+chr$(row+31)+chr$(col+31)" "; 'outline window
330 next col
350 next row
400 print esc$+"q";                        ' turn off reverse
410 print esc$+"H";                        ' home cursor within window
420 input "press return";a                 ' just to pause then do again
430 goto 100
```

The example shows in statement 320 that cursor addressing is relative to the screen and NOT the Window origins. The cursor positioning and the window definition ESCape sequences require that each co-ordinate be offset from 31 decimal or 32 decimal depending upon whether the application is using an origin of 1 or 0 respectively. This example uses an origin of 1, 1.

Note: The last line in this example will scroll up!

Fonts

The ROM BIOS has an in-built character Font limited to the first 128 characters in the standard Ascii set detailed in the appendices.
This permits the BOOT sequence to use the screen.

This is an 8 x 8 bit Font, i.e. 8 bytes per character, which drives the screen in 200 line mode.

The RAM BIOS subsequently loads the file "FONT.SYS" into the user RAM which consists of 2 different Fonts for the full 256 Ascii character set. They are:

1. 8 x 8 for 200 line mode
2. 8 x 10 for 256 line mode

The Pointers:

00706H - Base of Active Font
0070AH - Length of Active Font
0070CH - Base of Master Font
00710H - Length of Master Font
(i.e. of 8 x 8 and 8 x 10)

are then initialised to point to either Font. (See Appendix C). The 8 x 10 Font is located at Master Font + 2048 bytes (i.e 256 x 8 bytes).

The character Fonts in FONT.SYS may be altered by using the Fontedit utility supplied with the Apricot or alternatively by modifying the RAM based Font.

The latter may be achieved in two ways:

1. Loading a complete new table and changing the pointers to the active table to reflect the location and length. Refer to Keyboard table example.

   Note that the Master pointers are required by the screen driver to provide for environment switching. Care must be taken in changing them.

2. By modifying one or more locations directly in the existing Font.

Ascii control codes

The Screen Driver handles control codes i.e. codes with an Ascii value which is less than 20 hex as follows:

00	- NUL	: no action
01	- SOH	: no action
02	- STX	: no action
03	- ETX	: no action
04	- EOT	: no action
05	- ENQ	: no action
06	- ACK	: no action
07	- BEL	: sounds the BELL
08	- BS	: moves cursor back one space
09	- TAB	: moves cursor right in modulus of eight
0A	- LF	: moves cursor down 1 line
0B	- VT	: moves cursor down 1 line
0C	- FF	: moves cursor down 1 line
0D	- CR	: moves cursor to beginning of current line
0E	- SO	: no action
0F	- SI	: no action
10	- DLE	: no action
11	- DC1	: no action
12	- DC2	: no action
13	- DC3	: no action
14	- DC4	: no action
15	- NAK	: no action
16	- SYN	: no action
17	- ETB	: no action
18	- CAN	: no action
19	- EMM	: no action
1A	- SUB	: no action
1B	- ESC	: invokes an ESCape sequence
1C	- FS	: no action
1D	- GS	: no action
1E	- RS	: no action
1F	- US	: no action

ESCape Sequence Table

The table set out in this section lists the complete set of ESCape sequences for the Generic BIOS.

Appendix E lists the ESCape sequences by type of activity, e.g. WP primitives, Keyboard related, etc.

ESCape sequences consist of a sequence of characters always commencing with the ESCape code 27 decimal. This is best illustrated with BASIC as follows:-

```
PRINT CHR$(27)+"function"+"parameter(s)"
```

where:

function is an Ascii value in the range 20H to 7DH

parameter(s) is an Ascii value in the range 00H to 7FH

(The number of parameters varies according to the function).

ESCape sequences may have no meaning on certain machines. For example those affecting the Microscreen are not implemented throughout the Apricot family. Also certain ESCapes are listed as No-op's. Do not use these ESCape sequences - they are not supported.

The parameters supplied for cursor control sequences are based upon an origin of column 1 line 1. Any other required origin should be relative to the base of 32 decimal.

Availability is denoted by the Machine Key in each entry. The Key is:

A - Apricot
F - F1
P - Portable

ESCape sequence table

CHAR	HEX	KEY	Function
#	23		No-OP
$	24	PFA	Transmit Character Sends the character under the cursor into the keyboard buffer.
%	25		No-OP
&	26	PFA	Print Page Outputs the contents of the screen to the line printer. A form-feed is executed first.
'	27	PFA	Print line Outputs the entire line that the cursor is at to a connected line printer. No form feed is executed.
(	28	PFA	Set High intensity Mode Shadow-prints all characters to give the effect of high intensity characters.
)	29	PFA	Set Low intensity Mode Clears the mode set above.
*	2A	A	Change to second character font.
+	2B	A	Clear all high intensity characters.
,	2C	PFA	Set Window size. Takes four parameters in Ascii: <1> Top line + 31 <2> Bottom line + 31 <3> Left-hand column + 31 <4> Right-hand column + 31
—	2D	A	Clear all low-intensity characters.
.	2E	PFA	Reset Window Size. Resets the window size set by code 2C
/	2F	A	Set membrane key LED's.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
0	30	PFA	Sets underline mode. All characters printed have a single line of pixels placed under them to simulate underline.
1	31	PFA	Reset underline mode. Cancels the mode set above.
2	32	A	No-op
3	33	A	No-op

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
4	34	PFA	Change the representation of a key. This escape sequence takes 3 parameters. They are: <1> - Key mode (ascii) : 1 = normal 2 = shift 3 = control <2> - Key number — 1: e.g. 0 = help... etc. (see Appendix B) <3> - New key character : Ascii char. or hex equivalent. 10 PRINT CHR$(27)+"4"+"3"+CHR$(72) +"C" This example changes the key with the legend "C" (downcode number 72), in control mode, to generate an Ascii "C" (43H) as opposed to a binary 03H. The key has the default attribute of AUTO-REPEAT. Refer to Appendix B for a list of keys, their corresponding character value, down code and attributes. Note: The Screen Driver does not accept non-printable characters (range 0 to 31) in ESCape sequences therefore no Key or Key character can be programmed with a value below 32. See <2> and <3> above.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
5	35	PF	Set character foreground colour. This escape sequence takes 1 parameter which is from "0" (30H) to "?" (3FH). This gives 16 possible indexes. For a list of indexes and colours represented by them see escape sequence "]" (5DH). See also the diagram for ESC "6" (36H) below.
6	36	PF	Set block or background colour. This escape sequence sets the colour of all pixels in the character cell that do not make the actual character shape. As above, it takes 1 parameter. Diagram of a character cell: Background (all 0's) 00000000 00111100 00100100 001001 00111100 00100100 00100100 00000000 Foreground or text colour (all 1's)

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
7	37	PF	Set screen environment. This sequence is followed by one of the following mode parameters: "0" - Apricot PC & Xi monochrome compatibility "1" - 80 column full colour display "2" - Reserved "3" - 80 column Apricot compatible Display Refer to section Screen environment for operational details.
8	38	PFA	Set literal/Test mode ON The escape sequence tells the screen driver to perform the following action on receiving the next character: Ignore the fact that it is a control code (<20H) and print the character associated with it. This means that the font cell characters under (20H) are printed, rather than obeyed. The ESCape must be sent for each character.
9	39	PFA	Set strikeout mode ON All following characters are displayed with a horizontal line through the centre. This is widely used for deleting data within legal documents.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
:	3A	PFA	Set strikeout mode OFF. The code reverts the action of "9" (39H).
;	3B	PFA	Position cursor to start of status line. The status line is the 25th line of the Screen. The Cursor remains on this line until a position Cursor command is given.
<	3C	A	Display time on MSCREEN.
=	3D		No-OP
>	3E		No-OP
?	3F	A	Enter CALC mode This escape sequence switches on the internal BIOS calculator. It is the same as pressing the "Calc" key.
@	40	PFA	Enter insert mode. After this escape sequence is issued, whenever a character is printed, all the characters to the right of it will be shifted right one place and the character will be inserted in the space created.
A	41	PFA	Cursor UP.
B	42	PFA	Cursor DOWN.
C	43	PFA	Cursor RIGHT.
D	44	PFA	Cursor LEFT.
E	45	PFA	Clear screen. The current window is cleared, and the cursor is homed.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
F	46	PFA	Enter VT52 Graphics mode. VT52 mode displays the VT52 standard graphics characters represented by the Ascii value of Apricot lower case characters.
G	47	PFA	Exit VT52 Graphics mode. Invoke this mode to exit VT52. Failure to do this results in non lower case letters being incorrectly displayed.
H	48	PFA	Home Cursor. The cursor is placed at the top left hand corner of the current text window.
I	49	PFA	Reverse-index and line-feed. This sequence moves the cursor up one line. However if the cursor is at the top of a window then a scroll DOWN of the whole window is performed.
J	4A	PFA	Erase to end of Page. The sequence first of all erases all characters from the cursor position to the end of the current line, and then all subsequent lines below the cursor till the end of the current window or page.
K	4B	PFA	Erase to end of line. This escape sequence erases all characters from the cursor position to the end of the defined right-hand side margin.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
L	4C	PFA	Insert line. This sequence places the cursor to the beginning of the current line, and then inserts one line below the current cursor position by scrolling all subsequent lines down by one place.
M	4D	PFA	Delete line. This sequence places the cursor to the beginning of the current line and scrolls all lines under it up by one place.
N	4E	PFA	Delete character. The character under the cursor is cleared, and all characters to the right are scrolled left by one position. This is active in the defined right-hand margin space.
O	4F	PFA	Exit Insert Mode. This sequence reverses the effect of ESC "@" (40H)
P	50	PFA	Insert single character. This sequence scrolls all characters from the current cursor position to the defined right-hand margin right by one place.
Q	51	PFA	Scroll left. Takes one parameter which is the number of columns plus 31 that the screen is to be scrolled. This is only active in the current window.
R	52	PFA	Scroll right. As above but scrolling left.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
S	53	PFA	Scroll up.
			As above, but scrolling is up.
T	54	PFA	Scroll down.
			As above, but scrolling is down.
U	55	A	Enable dual-output to MSCREEN
V	56	A	Disable dual output to MSCREEN.
W	57	A	Output text to MSCREEN only.
X	58		No-OP.
Y	59	PFA	Position Cursor.
			This sequence takes two parameters. They are the line number and column number in normalised Ascii.
			e.g. PRINT CHR$(27) "Y" CHR$(10+31) CHR$(15+31)
			will position the cursor at line 10, column 15.
Z	5A	PFA	Identify as VT52.
			This escape sequence is included as most of the screen driver is DEC VT52 compatible. After issuing this sequence the keyboard buffer is filled with three characters which can be read by an application to determine the device type.
[	5B	PFA	ANSI lead-in character.
			Refer to section ANSI ESCape sequences for details of the ANSI codes supported.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
/	5C	PFA	Place key in keyboard buffer. This sequence takes one parameter which is placed in the keyboard buffer. If the buffer is full the bell will sound and the character will be ignored: e.g. PRINT CHR$(27) "/R" will place an "R" into the keyboard buffer.
]	5D	PF	Set palette code. This escape sequence takes two arguments. The first is the index which needs to be changed, and the second is the colour. e.g. PRINT CHR$(27) "]05" sets index 0 (in this case the background) to colour 5. Refer to the Colour section for details of index, colour and default palette settings.
^	5E		No-OP
_	5F		No-OP
`	60	PFA	Save environment The first three environment flags are saved. They can then be temporarily changed and restored by another sequence.
a	61	PFA	Restore environment Returns the first three environment flags to their state just after code 60.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
b	62	PFA	Erase from start of page All characters from the top left-hand corner of the current defined display page size to the current cursor position are cleared.
c	63	A	Disable MSCREEN scrolling.
d	64	A	Enable MSCREEN scrolling.
e	65	A	Switch MSCREEN cursor ON.
f	66	A	Switch MSCREEN cursor OFF.
g	67	A	Disable Time and Date display on microscreen.
h	68	PFA	Reverse tab This sequence has the opposite effect of control code 9 hex, it performs a tabulation operation to the left rather than the right.
i	69		No-OP
j	6A	PFA	Save cursor position The current cursor position is noted within the BIOS.
k	6B	PFA	Restore cursor The cursor is restored to the position it was in when ESC 'j' was executed.
l	6C	PFA	Erase line The line which the cursor is on is cleared. Note that no scrolling takes place
m	6D	PFA	No-OP

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
n	6E	PFA	Return Cursor position. The current cursor position is placed into the keyboard buffer in the ESC+"Y" format with a base column and row of 1.
o	6F	PFA	Erase to start of current line. All characters from the start of the current line up to and including the character under the cursor are cleared.
p	70	PFA	Enter reverse video mode. All characters printed after this sequence are displayed in inverse video.
q	71	PFA	Cancel inverse video mode. Restores the writing mode to the state it was in when ESC 'p' was executed.
r	72	A	MSCREEN echo enable.
s	73	A	MSCREEN echo disable.
t	74		No-OP
u	75		No-OP
v	76	PFA	Wrap at end of line. This escape sequence indicates that the normal screen driver action when the cursor reaches the end of a line should be employed. The action is to return the cursor to the beginning of the next line on the screen.

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
w	77	PFA	Discard at end of line. When the cursor reaches the end of the current line, it will remain there and all characters are printed under it.
x	78	PFA	Set environment flags. Takes one parameter:
		PF	G - reset screen palette
		PF	$ - set to Apricot Compatible mode on default screen
			1 - enable line 25
			2 - nothing
			3 - nothing
			4 - nothing
			5 - cursor off
			6 - nothing
			7 - nothing
			8 - set auto LF on receipt of CR
			9 - set auto CR on receipt of LF
			A - nothing
			B - nothing
			C - nothing
		PF	D - smooth screen scrolling
		P	E - LCD contrast up
		PF	F - bell volume down

ESCape sequence table (continued)

CHAR	HEX	KEY	Function
y	79	PFA	Reset environment flags. Takes one parameter: 1 - disable line 25 2 - nothing 3 - nothing 4 - nothing 5 - cursor on 6 - nothing 7 - nothing 8 - no auto LF on receipt of CR 9 - no auto CR on receipt of LF A - nothing B - nothing C - nothing D - fast screen scrolling E - LCD contrast down F - bell volume down
{	7B	PF P PF	No-OP
|	7C		No-OP
]	7D	PF	Reserved for GSX - call is not supported.

Screen Bit Image

The Apricot Portable has a bit screen memory which enables normal text and graphics to be combined on the same screen.

The Bit Image is fully discussed both from a physical and a software point of view in the Hardware section of this manual.

The Screen Driver provides the Apricot with a choice of environments as discussed in the Applications section.

The block diagram below shows how the Bit Image is logically divided to support the Apricot in the Screen Driver environment.

The minimum configuration has memory at location 0E0000 thru 0EFFFF (word wide) only. The Driver supports either the LCD in Apricot compatible mode or the Display in 4 from 16 colours.

A further option is available to add RAM from 0D0000 thru 0DFFFF to provide 16 Colour support. However the Screen driver only supports 8 from 16 colours in the maximum configuration in order to accomodate the LCD. The diagram below depicts the Bit Image layout.

The memory is logically split into 4 Planes each of 32K bytes.

In 4 from 16 mode the LCD is turned OFF. The Screen Driver uses Planes 3 and 4 to hold the Display image and to derive the 4 colours.

Planes 1, 2 & 4 are used by the Screen driver to derive the 8 available colours while Plane 3 is reserved for the LCD. The Screen driver programs the palette to ignore data (i.e. programs it to the background colour) from Plane 3 as described in the Hardware section.

Planes 1, 2, 3 and 4 may be used by Applications to derive all 16 colours of the palette. However it can be seen that in this case the LCD cannot be supported. The Screen driver does not currently support this mode.

Keyboard driver 3.4

Contents

Illustrations

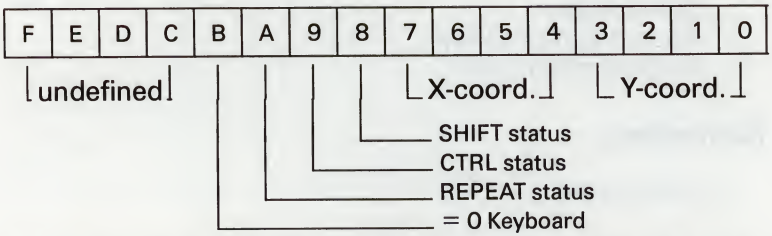
Overview

The Keyboard driver software handles communications between the SIO and MS-DOS. Serial data is received by the SIO from both the Keyboard and the Mouse.

The driver distinguishes between the two and vectors to the appropriate routine. The Mouse driver is of sufficient importance to be treated as a separate section of the drivers. Here we will concentrate on the keyboard.

Applications running under MS-DOS receive data from the keyboard via normal MS-DOS CALLS (INT 21H).

The action of pressing a key or combination of keys, e.g. SHIFT+ or CTRL+, will result in the SIO generating an interrupt and passing a packet of data in Hamming coded format to the SIO interrupt handler. The data is then converted and presented to the driver in the AX register in the following format:



A similar packet is sent for the Mouse except that bit B = 1.

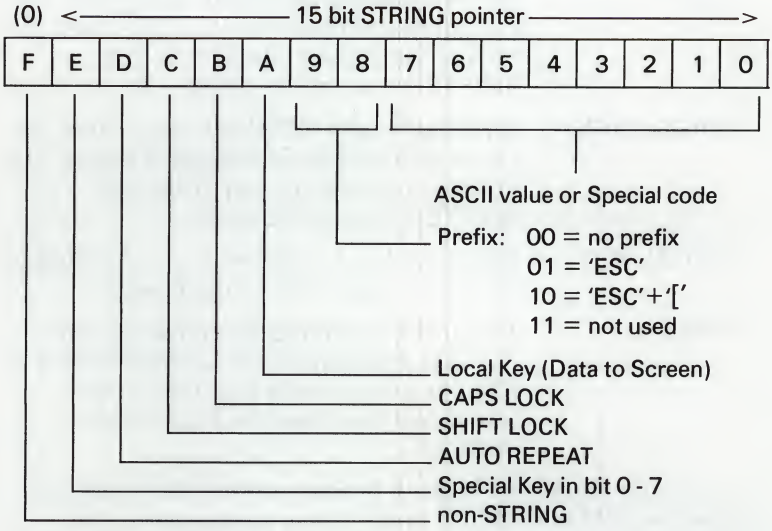
The X and Y coordinates are converted into an offset within the soft keyboard table. The generic keyboard has 104 keys and the offset is in the range 0 to 103. Each of these is referred to as a "down-code".

The keyboard table is divided into sections, each of 104 words, to facilitate the following:

Normal - key without SHIFT or CTRL
Shifted - SHIFT + key
Control - CTRL + key

The attribute bits 8 and 9 of the packet determines which section of the table is to be used to translate the down-code.

The translation of a down-code is dependent upon further attributes in the corresponding entry of the table. Each entry is a word of the following format:



Attribute bits A - F are: 'ON' if set to 1
'OFF' if set to 0.

The exception to this format is when "Non-STRING" attribute bit F is set 'OFF', implying that the down-code is to be translated into a string of data. In this case the format is:

bit 0 - bit E - String offset of 15 bits (i.e. 32K)
bit F - set 0 = STRING

The STRING facility enables any key to be translated into a string of data. e.g. The F1 key may be interpreted in BASIC as the character string RUN plus a CR which results in 4 characters being returned to the application level via MS-DOS.

These STRINGS, which may be of variable length, are appended to the end of the three sections of the Keyboard table described above. The default table is 1024 bytes long - the first 312 words for each of the 3 sections described above, followed by a 200 word table reserved for "Default STRINGS".

The remaining attribute bits have the following meaning:

AUTO REPEAT	When set enables auto repeat on this key.
SHIFT LOCK	When set in the NORMAL entry then the Key is affected by SHIFT LOCK mode. The driver translates the Key from the SHIFTED area of the table.
CAPS LOCK	When set in the NORMAL entry then the Key is affected by CAPS LOCK mode. The driver translates the Key from the SHIFTED area of the table.
LOCAL KEY	When set the translated key is directed to the screen and NOT to the Queue.
PREFIX	The prefix is derived from the value of two bits and dependent upon the value an ESCape sequence is pre-fixed to the translated data (see the section on prefixes).

The default Keyboard Table, if present on the Boot disk, is loaded into RAM at BOOT time. It may subsequently be modified either by replacing it entirely with another table or simply by changing specific entries. This procedure is described in the section Applications Interest.

The Driver translates each key into one or more data bytes which are then appended to a 'ring buffer' queue. If the data will not fit on the end of the queue then it is lost and the BELL is sounded.

MS-DOS accesses the keyboard via the RAM BIOS using calls to the Control Device: typically requests for "get data from queue" or "Flush queue".

Application programs may by-pass MS-DOS and use the same facilities in the Control Device.

This is of advantage where the normal operation of the keyboard driver is not required. For example, if the translation procedure described above is not required, the Application may alter the actions of the keyboard driver to return the down-codes in the queue, without translation. These down-codes are referred to as "Raw data". Various other facilities are also available and detailed in the Control Device chapter.

The following section describes with examples how the applications may take advantage of the Generic keyboard facilities. In addition, it describes in detail all of the Special Keys and their actions.

The Systems interest section describes the modules making up the driver and special features implemented in the software such as Auto-repeat.

Applications Interest

Changing the keyboard table

There are three methods of changing the Keyboard table in RAM, they are:

1. Use the system utilities to modify the system table.
2. Load a new table and change pointers to it.
3. Change values within the default table.

If the modification is to be permanent then the Keyedit Utilities should be used to amend the Keyboard table.

To implement the second option above it is sufficient to load the new table into a reserved area of memory and then to change the Pointer to the "active keyboard table" to reflect the new location. REMEMBER to restore the original pointer if the following Application is not known.

Example: Load a new keyboard table and point to it.

```
100 DIM TABLE%(1023)                          'data area for user Table
110 TABOFF%=VARPTR(TABLE%(0))                 'offset within data segment

120 CALL GETSEG%(SEG%)                        'data segment (refer Appendix F)

200 DEF SEG=0                                 'segment for PEEKS & POKES
210 KS=PEEK(&H0714)+(256*PEEK(&H0715))        'segment - old pointer
220 KO=PEEK(&H0712)+(256*PEEK(&H0713))        'offset - old pointer

240 POKE &H714,(SEG% AND &HFF)                'segment - new pointer
250 POKE &H715,(SEG% AND &HFF00)/256
260 POKE &H712,(TABOFF% AND &HFF)             'offset - new pointer
270 POKE &H713,(TABOFF% AND &HFF00)/256

300 DEF SEG                                   'basic data segment
310 BLOAD "KEYTAB",TABOFF%                    'load new table
```

The memory image of the table in "KEYTAB", which must have been created with a BSAVE statement, is loaded into the array TABLE%. The called subroutine at GETSEG% is defined in Appendix F (Language interfaces to the BIOS). This routine must be merged with the example program for execution purposes.

Statements 210 and 220 simply save the BIOS pointers to the active keyboard table in order that they may be restored after execution of the program.

To change values within the default table simply pick up the Pointer to the base of the table and modify the specific locations offset to the base address.

The example of switching keyboard tables above is dependent upon a table being previously BSAVE'd. Many other methods of loading a Keyboard table may be derived and in certain cases will be necessary. Compiled Microsoft Basic does not support BSAVE, for example, so a different method is required.

The BLOAD and BSAVE method, however, serves the purpose of illustrating the technique and for completeness the example below shows how to save a table.

Example: Save the active keyboard table

```
120 CALL GETSEG%(SEG%)                          'data segment (refer Appendix F)

200 DEF SEG=0                                   'segment for PEEKS & POKES
210 KS=PEEK(&H0714)+(256*PEEK(&H0715))          'segment - old pointer
220 KO=PEEK(&H0712)+(256*PEEK(&H0713))          'offset - old pointer
230 KL=PEEK(&H0716)+(256*PEEK(&H0717))          'length

300 BSAVE "KEYTAB",KO,KL                        'save active table
```

This example may be brought into perspective by combining it with the previous example and the one that follows in the next section. The result is a Keyboard Table being saved with a few changes from the default. Then the modified table may be loaded at any time as described above.

Minor changes to the Keyboard Table at run time may also be made by simply picking up the Pointer to the base of the table and modify the specific locations offset to the base address.

Alternatively further techniques are described in the Screen Driver chapter in the ESCape sequence section.

Implementing Key STRINGS

There are many cases in practice where the depression of a single key can eliminate the tedium of typing in a complete sequence of characters. In BASIC certain verbs are commonly used in the interactive mode while testing, for example:

PRINT - to check contents of variables
LOAD - to load files
SAVE - to save files

Whatever the application, keys may be modified to reflect the input of a complete string of characters to the keyboard. Typically the function keys would be used to accomplish this.

The Key STRINGS are located at the end of the 3 sections of the down-code translation table. In order to add Key strings to the list it is neccessary to perform the following:

1. Insert the new Key STRING and attributes at the end of any existing assignments.
2. Modify the down-code table to point to it.

The following program illustrates how to modify the 9 function keys F1 - F9 to represent BASIC commands as follows:

F1 - RUN	F6 - PRINT
F2 - AUTO	F7 - LOAD
F3 - LIST	F8 - SAVE
F4 - LLIST	F9 - DELETE
F5 - RENUM	

In addition F1 to F5 are set up to generate the STRING + a carriage return when SHIFT+Fkey are pressed.

The program sets up the following parameters in DATA statements:

MODE - 0 = normal, 1 = SHIFT, 2 = CONTROL
KEY - down-code
KEY$ - STRING
CR - 0 = no CR, 1 = with CR

It searches from the end of the table for the first free STRING entry and then places each new entry successively at the end of the table and updates the pointer in the relevant part of the down-code table. If the CR parameter is non-zero then an additional entry is made in the SHIFT section with the addition of a CR (Carriage return).

Example: Implementing Key Strings

```
100 REM set up function keys
110 DATA 9 'no of items in list
120 DATA 0,0,"RUN ",1
130 DATA 0,1,"AUTO ",1
140 DATA 0,2,"LIST ",1
150 DATA 0,3,"LLIST ",1
160 DATA 0,96,"RENUM ",1
170 DATA 0,4,"PRINT ",0
180 DATA 0,5,"LOAD ",0
190 DATA 0,6,"SAVE ",0
200 DATA 0,7,"DELETE ",0

500 REM get pointers
510 DEF SEG=0
520 KS=PEEK(&H0714)+(256*PEEK(&H0715)) 'segment
530 KO=PEEK(&H0712)+(256*PEEK(&H0713)) 'offset
540 KL=PEEK(&H0716)+(256*PEEK(&H0717)) 'length

600 DEF SEG=KS
610 J=KO+KL-1                                    'end of table
620 WHILE PEEK(J)=0 : J=J-1 : WEND
630 J=J+1 'start of free string table
635 REM "check for sufficient space !!! "

640 READ N 'no of DATA statements
650 FOR I=1 TO N : READ MODE,KEY,KEY$,CR

660 POKE J,LEN(KEY$) 'STRING length
670 POKE J+1,0        '          attrib
680 FOR X=1 TO LEN(KEY$)
690 POKE J+1+X,ASC(MID$(KEY$,X))
700 NEXT X

720 DEF SEG 'BASIC data
730 P%=J 'set up 15 bit pointer
740 P1=PEEK(VARPTR(P%)):P2=PEEK(VARPTR(P%)+1)
750 DEF SEG=KS 'put it in down-code table
760 POKE KO+(MODE*208)+(KEY*2),P1
770 POKE KO+(MODE*208)+(KEY*2)+1,P2

780 J=J+2+LEN(KEY$)+1 'point to next free entry
790 IF CR<>0 THEN KEY$=KEY$+CHR$(13): MODE=CR: CR=0 : GOTO 660
800 NEXT I
```

Note: The example may modify the default table settings which already have pre-defined functions such as invoking the calculator or controlling the LCD. Refer to the Keyboard Table in the Appendix for details.

Changing the Keyboard driver operation

The Overview of this chapter describes how the Keyboard driver works by default.

The mode of operation may be changed by "special" keys, which are detailed in a later section, or by calls to the Control device. The latter is considered below.

The Keyboard driver normally translates down-codes into data from the active keyboard table and then places the data in the queue.

Some applications, however, require that the "raw" down-codes are not translated. This may be achieved by changing the "Fall-thru" mode via command 0004H of the Control device.

Example: The "Fall-thru" mode

```
100 DEF SEG=60H : IO=0                          'point to Control device vector
105 COM%=7:DAT%=0:GOSUB 1000                    'set HELP ignore off
110 COM%=4:GOSUB 2000                           'toggle the "Fall-thru" switch
120 IF RET%=1 THEN PRINT "RAW Down-codes:"
130 IF RET%=0 THEN PRINT "ASCII normal values"
200 COM%=&HB:gosub 1000                         'wait for char. in queue
210 PRINT RET%;                                 'print its value
220 IF RET%=1 THEN 110                          'if true then toggle mode
230 GOTO 200
1000 DEV%=&H32
1010 CALL IO(DEV%,COM%,DAT%,RET%)               'execute command
1020 RETURN
2000 DAT%=-1: GOSUB 1000                        'get current setting
2010 DAT%=RET% XOR 1: GOSUB 1000                'toggle it
2020 RETURN
```

The example displays the current "Fall-thru" setting and waits for a key to be pressed which results in data being placed in the queue.

The options are to display the down-code in the range 1 - 104 or to display the translated value of the key from one of the three sections of the table, i.e. Normal, SHIFT or CTRL.

The program toggles the "Fall-thru" mode whenever it receives the value of 1 in RET%. This value is the result of different key depressions in the two modes, they are:

HELP (or F1) - in "Fall-thru"
CTRL+A - in normal mode

The HELP key has special significance to the driver and if it were detected in this program then it would not work. For this reason the HELP ignore call is made at the beginning of the program. Refer to Systems interest for further details.

The subroutine at statement 2000 illustrates the GET/SET feature of the Control Device. The setting of invalid data in line 2000 results in the call returning the current setting unchanged. Line 2010 then toggles the setting with an exclusive OR and calls the Control Device again.

Refer to the Control Device for further calls affecting Keyboard driver operation.

Notes:

1. The down-codes presented to the driver are in fact in the range 0 to 103. However the driver increments "raw" data to generate a range 1 to 104 before placing it in the queue.
2. In the ASCII mode, keys which are designated as "STRINGS" will be actioned and not displayed. This means that the corresponding string will be placed in the queue.

Special Keys

Certain keys in the down-code keyboard table are designated special keys by setting attribute bit E hex to 1. In these cases bits 0 - 7 specify the key type. These types have a specific meaning to the Keyboard driver i.e. they act as switches. Each type by default is associated with a key in one or more of the normal, SHIFT and CTRL. The types and their respective meanings are as follows:

00 reserved for spare keys - the driver takes no action when these are encountered.

01 this type toggles the SHIFT LOCK and CAPS LOCK driver modes and their respective LED's. i.e.

- if in CAPS LOCK then clear lock and LED
- if in SHIFT LOCK then clear lock and LED
- if in CONTROL mode set SHIFT LOCK and LED
- if in NORMAL mode set CAPS LOCK and LED

02 no action on Apricot F1 and Portable

03 no action on Apricot F1 and Portable

04 no action on Apricot F1 and Portable

05 sets/resets the STOP mode and LED

06 sets/resets CALCULATOR mode and sends start/stop sequences to CALCULATOR.

07 this generates an F7 hex VOICE interrupt.

Refer to the default Keyboard table in Appendix B for down-code designations of these keys. Under normal circumstances these should not be changed.

The generic control device may also be used to invoke these functions via software control. Refer to the Keyboard driver section and the command "Get/Set Keyboard status".

Default STRINGS

The default keyboard table has a number of pre-defined STRINGS. These are invoked by special key combinations. The STRINGS are located immediately after the end of the 3 sections of the down-code table.

These strings may be freely changed or re-designated as required by the application. Note that the Example program to add STRINGS to the list in an earlier section searched from the end of the allocated table space until it found the end of these default STRINGS. It does not need to know exactly where or what their size is.

The Default strings have been implemented to provide extra facilities for the Keyboard, and the colour monitor.

The keys and their functions for the Portable are:

CTRL + F1	- 80 column Apricot compatible mode on the LCD
CTRL + F2	- 80 column multi-colour
CTRL + F3	- 40 column mode on the LCD
CTRL + F4	- 80 column Apricot compatible on the Display
CTRL + UP arrow	- LCD contrast up
CTRL + DN arrow	- LCD contrast down
CTRL + R arrow	- Bell volume up
CTRL + L arrow	- Bell volume down
SHIFT + UP arrow	- Fast screen scrolling
SHIFT + DN arrow	- Smooth screen scrolling

The keys and their functions on the F1 are:

CTRL + F1	- 80 column Apricot compatible mode
CTRL + F2	- 80 column multi-colour
CTRL + F3	- 40 column mode
CTRL + F4	- as CTRL + F1
CTRL + R arrow	- Bell volume up
CTRL + L arrow	- Bell volume down
SHIFT + UP arrow	- Fast screen scrolling
SHIFT + DN arrow	- Smooth screen scrolling

Prefixes

Each entry in the down-code table has a facility to prefix the data value of the key with an ESCape sequence. The two bit prefix in the attributes has the following meaning:

00 - no prefix

01 - prefix with 'ESC' this facility is used with screen control keys such as HOME, CLEAR and the arrow keys. The character 1B hex (27 decimal) is prefixed to a data character and placed in the queue.

10 - prefix with 'ESC'+'['

'ESC' and '[' prefix an ANSI sequence.

11 - is not used

Refer to the Keyboard Table in the Appendix for more details.

User Interrupt (F9 hex)

The Keyboard User Interrupt (F9 hex) is invoked by the driver prior to placing the decoded key character(s) in the Receive Queue.

A copy of the data in the AX and BX register is passed to the user interrupt with the following format:

Low byte = Received character (translated)

High byte = Count of the number of chars. This will be 1 for non-STRING keys. For STRING keys it will be a count of the number of characters to expect inclusive of the current low byte.

The User routine, unless otherwise initialised by the Application, is a dummy routine which will return BX unchanged and AX = 0FFFF hex.

Applications may define a User interrupt to replace the dummy routine. In particular this is useful for the handling of "raw" down code data.

If the routine returns AX = 0FFFF hex then the Driver places the contents of register BL in the Queue.

If the routine returns AX = 0000 hex then the character is ignored, i.e. thrown away.

Any translation or manipulation of the characters may take place and be returned to the Keyboard driver in the BL register for subsequent addition to the queue.

Systems interest

The Keyboard driver consists of various modules which include the following functions:

1. Initialisation
2. Steering

   clock set-up
   time and date entry
   raw code handling
3. Down-code handling
4. Queue handling
5. Configuration

The drivers are not directly accessible by applications software. The correct use of them should always be via the Control device.

In the following sections, each module is discussed to give an overview of internal operation and to provide more detail about certain Control device calls.

Initialisation

This module is executed by the Boot process and performs the following:

zeroises and initialises queue

initialises SIO and keeps copy

Note: The Control device call to initialise only initialises the queue.

Steering

This module performs the function of receiving the data from the SIO and filtering it to the appropriate routine. The following steps are taken:

1. Is the data a Mouse packet?
   Execute the Mouse interrupt 3. This interrupt handles the Mouse packet completely.
2. SET TIME key pressed?
   If the data is the SET TIME key then the 25th line of the display is enabled in calculator mode. The Time/Date prompt is displayed and the next 10 characters of input are assumed to be in the format HH MM DD MM YY (with no spaces in between). The clock within the keyboard is set to this time.
3. TIME/DATE key pressed?
   This key resets the clock implemented by the clock driver with the current setting of the keyboard clock.
4. KB LOCK pressed?
   This key enables the user to lock the keyboard to prevent accidental access. The key has a simple toggle action; lock on, lock off. The driver also controls the corresponding LED on the front panel according to the LOCK status.
5. Normal XY keycode?
   If the LOCK is on then ignore any key. If the XY code is 80 hex or greater then it is ignored. The XY code is translated into a down-code in the range 0 to 103. The routine to handle normal keyboard codes is called.
6. RAW mode active?
   If Fall-thru mode is on then the down-code is adjusted to the range 1 to 104 and added to the queue.
7. HELP key pressed?
   If in RAW mode and the HELP ignore status is 'off' then toggle reset the RAW mode off.

Down-code handler

This is the principle module of the Keyboard driver. It is broken down into a number of sections.

The down-code is passed to the handler in the AX register with the status in the high byte and the down-code itself (with a value derived from the XY coordinates in the range 0 to 103), in the low byte.

Firstly the down-code is used as an offset into one of the three sections of the active Keyboard table to produce one of the following words:

DATA	where bit 0 - 7	= ASCII key data
	bit 8 - 9	= prefix 00 none
		01 'ESC'
		10 'ESC'+'['
		11 'ESC'+'O'
	bit A	= Local
	bit B	= CAPS LOCK
	bit C	= SHIFT LOCK
	bit D	= AUTO REPEAT
	bit E	= 0 - not SPECIAL
	bit F	= 1 - not STRING
SPECIAL	where bit 0 - 7	= type of special key
	bit 8 - 9	= 00 - no prefix
	bit A - D	= 0000 - ignored
	bit E	= 1 - SPECIAL
	bit F	= 1 - Non-string
STRING	where bit 0 - E	= 15 bit string pointer
	bit F	= 0

If the CAPS/SHIFT LOCK mode is set on then down-codes received in NORMAL mode whose corresponding entry has CAPS/SHIFT LOCK enabled are vectored to the SHIFTed area of the table.

Refer to the overview in this chapter for a definition of the bit settings within each word.

The following sequence then takes place within the down-code handler.

1. SPECIAL keys are filtered and processed to affect keyboard status and LEDS as follows:

CAPS LOCK	if in CAPS LOCK clear lock and LED if in SHIFT LOCK clear lock and LED if in CTRL mode set SHIFT LOCK/LED if normal mode set CAPS LOCK/LED
STOP	sets/resets keyboard status and LED
CALC	sets/resets keyboard status sends start/stop sequences to calculator
VOICE	executes an F7 hex software interrupt

2. STRING and DATA keys are filtered and sent to the destination prefixed as neccessary. The only difference being that the for DATA only one character is sent.

   The 15 bit string pointer is to a location in the table with the following packet of data in bytes:

   Length (in bytes 'n')

   Type (as in normal entry bits 8-15)

   'n' DATA bytes

If the Local bit is set then the data is sent directly to the screen driver. If the screen driver is not active then the data is ignored.

Queues

The Keyboard I/O queue is an 80 byte circular buffer.

The Keyboard down-code handler passes the data generated by each relevant key to the Queue handler.

The Queue handler performs an interrupt F9 hex, which is described in the section on User interrupts on a previous page.

If there is insufficient room in the Queue then the BELL is sounded and the data is ignored.

The Queue may be accessed directly through the Control device.

The module consists of three distinct areas, they are:

1. Queue filler which adds one or more characters to the Queue. This area is called by the down-code handler and may also be called via commands 8 and D hex of the Control device.
2. The "look-ahead" facility which will return the next output character from the Queue without updating the I/O pointers. Command C hex.
3. The Queue reader which returns one or more characters from the Queue. Commands B hex and E hex.

Configurator

This module facilitates modification of the Keyboard Status and hence the way in which the Driver operates.

The Status may be altered directly by the Keyboard driver on receipt of certain keys or it may be programmed via the Control Device command F hex.

The Status byte has the following settings (bit = 0, is OFF, bit = 1 is ON):

bit 0 = Ignore HELP key in Fall-thru mode
bit 1 = STOP
bit 2 = CALC
bit 3 = SHIFT LOCK
bit 4 = Fall-thru
bit 5 = VOICE LED (On Portable only)
bit 6 = LOCK
bit 7 = CAPS LOCK

The Control Device may be used to both interrogate the Status and set it, simultaneously if necessary, by ANDing the byte with a specific mask to determine the current settings (typically with FF hex to return all status bits) and ORing with another byte to reset Status bits

Apricot compatibility

The Keyboard table format, function and handling is exactly the same on the generic range of Apricot Hardware.

The default size of the table is 1024 bytes divided into 3 sections, i.e. Normal, Shifted and CTRL; each of 104 words with the remaining space being reserved for string keys.

The key numbers are the same with the following qualifications:

1. The function keys F1 to F4, F6 to F10 are mapped onto the original Apricot pc/xi fixed function keys HELP ... FINISH (i.e. by legend rather than number. In addition the 97th entry is for F5 (VOICE).
2. In the default keyboard tables the function keys produce the following codes:

Normal	F1 (HELP)	= 177
	F2 (UNDO)	= 178
	F3 (REPEAT)	= 179
	F4 (CALC)	= 180
	F5 (VOICE)	= 185
	F6 (PRINT)	= 181
	F7 (INT)	= 182
	F8 (MENU)	= 183
	F9 (FINISH)	= 184
SHIFT:	F4 (CALC)	activates the calculator
	F5 (VOICE)	activates VOICE on the Portable
	F6 (PRINT)	activates hardcopy
CTRL	F1 (HELP)	80 Col Apricot compatible (LCD on the Portable)
	F2 (UNDO)	80 Col colour
	F3 (REPEAT)	40 Col mode
	F4 (CALC)	80 Col Apricot compatible (Display on the Portable)

The Membrane Keys (MicroScreen function keys) and their respective entries in the table are not used.

The special keys DATE/TIME, SET TIME, KEYBOARD LOCK, REPEAT RATE & RESET do not appear in the table. These are filtered out from the XY codes passed from the keyboard.

Certain "Special Keys" are no longer supported.

They are:

02 - right hand shift
03 - left hand shift
04 - Control Key
07 - Microscreen echo toggle

One additional "Special Key" is supported, i.e.

08 - VOICE (generates interrupt F7 hex)

The default string table for the Apricot F1/Portable is detailed in the section "Default strings".

Despite these differences the standard Apricot Keyboard Table may be used on the Portable and F1.

Configuration table

The section of the Configuration table reserved for the Keyboard Driver is located at displacement 10 hex in the table. It contains the following

Offset	Function
00	Key click volume (range 0 to F hex) 0 = full/F = off
01	Auto-repeat master enabler 0 = off/1 = on
02	Auto-repeat lead-in delay 1 to 255 times 20 ms
03	Auto-repeat interval of 1 to 255 times 20 ms
04	Microscreen mode (0 = time & date) (1 = screen echo)
05 - 0F	spare

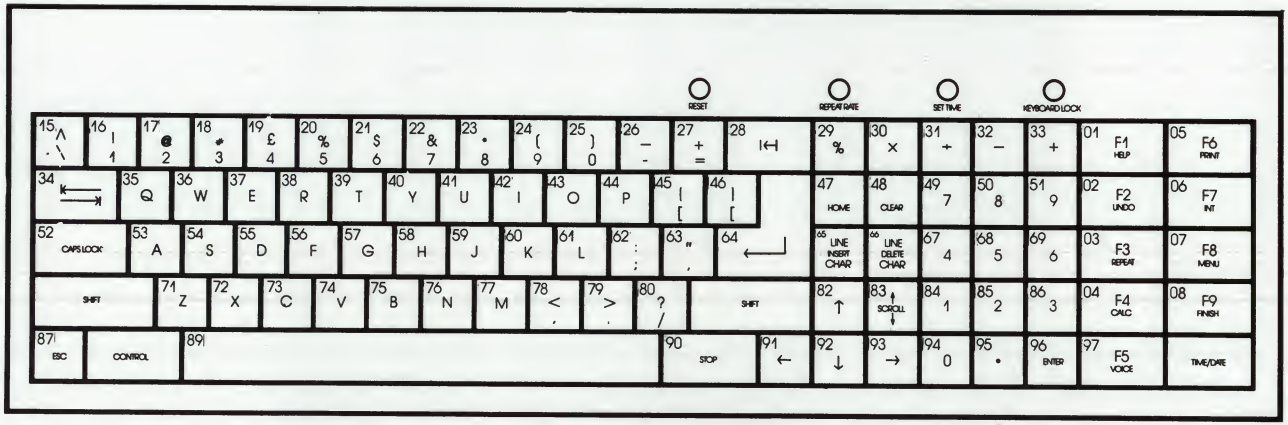


Figure 1. Downcodes

Serial I/O Driver 3.5

Contents

Overview

The Apricot Generic Serial I/O Driver provides asynchronous communications support for the following devices:

RS-232
MOUSE

The Control Device interface provides facilities to configure the Driver for different transmission speeds, line protocols etc.

Further, a call to the Control Device enables the Application, in conjunction with User interrupt (F0 hex), to switch control of the RS-232 channel from the BIOS to itself. This is described in the section on User interrupts.

The I/O port addresses and structure vary throughout the Apricot range but this is transparent to Applications using the Control Device. For Applications which access the hardware directly the above call to the Control Device provides details of the addresses and structure.

The RS-232 channel is switchable in the driver between Serial Mouse and RS-232 communications.

The Serial Mouse, when enabled, is handled by User Interrupt (FA hex). Details are described in the Mouse driver booklet.

The Applications section of this chapter details how to configure the Serial I/O Driver to the required Device and operational environment. The Systems interest section provides details of the configuration data governing the Serial Driver.

In RS-232 mode the Serial Driver may be configured through the Control Device calls to provide the following application environments:

Exclusive User interrupt control (F0 hex)

BIOS control which includes:
XON/XOFF protocol
RTS/CTS protocol
DTR/DSR protocol

Applications which require transparent (i.e non-BIOS) handling of the SIO are considered in the section "User Interrupts". Other applications, which are under BIOS control are discussed in the next section Configuring the Serial Driver.

Applications interest

Configuring the Serial Driver

The calls to the Control Device provide the Application with comprehensive facilities to configure the SIO serial interface in terms of protocol, transmission speeds, data format, etc., for asynchronous communications.

They also provide direct access to the control and status lines of the SIO to enable the application to use any variation, as required, of the standard line protocols provided. The requirement for such direct access is inherent in the field of communications. So many variations occur in external devices and their protocol as to exclude them from the scope of this chapter.

A common requirement in this field however is support of a Serial printer. The Apricot generic BIOS is usually configured to default printer output to the parallel Centronics interface. The example given below demonstrates how to configure the BIOS and switch to Serial printer communications.

The BIOS may be configured to Boot with the printer output directed to the Serial port. The Label Sector must be set to specify a Serial printer and the configuration data table must be set to the required transmission characteristics of the printer.

Alternatively in order to switch an existing Application during Run time to use a Serial printer, the Serial Driver must be configured to the requirements of the individual printer using calls to the Control Device and then a further call to the Control Device to disable parallel support and enable serial.

The example provided below illustrates this procedure for a printer with XON/XOFF protocol and the following option settings:

Switchable XMIT/RCVE baud rates and data bits
Stop bits (1, 1.5 or 2)
Parity (none,odd, or even)

Example: Configuring a Serial printer

```
 10 CLS$=CHR$(27)+"E"    'clear screen escape sequence
 20 DATA "NONE","ODD","EVEN"
 30 FOR I=1 to 3: READ PTY$(I): NEXT I
 40 DATA 1,1.5,2
 50 FOR I= 1 TO 3 : READ STOB(I) : NEXT I
 60 DATA 50,75,110,134.5,150
 70 DATA 300,600,1200,1800,2400
 80 DATA 3600,4800,7200,9600,19200
 90 DIM BAUD(15): FOR I=1 TO 15 : READ BAUD(I) : NEXT I
100 PRINT CLS$;"General data:"
110 INPUT " stop bits (1-3): "; RTS%
120 INPUT " parity (0-2): "; RTP%
130 PRINT

170 PRINT "XMIT parameters: "
180 INPUT " baud rate (1-15) : "; TXB%
190 INPUT " data bits (5-8): "; TXD%
200 PRINT

240 PRINT "RCVE parameters:"
250 INPUT " baud rate (1-15) : "; RXB%
260 INPUT " data bits (5-8): "; RXD%

300 DEV%=&H34                                  'Serial I/O Driver
310 COM%=4:DAT%=TXB%:GOSUB 900                 'set xmit baud
320 COM%=5:DAT%=RXB%:GOSUB 900                 'set rcve baud
330 COM%=6:DAT%=TXD%:GOSUB 900                 'set xmit data bits
340 COM%=7:DAT%=RXD%:GOSUB 900                 'set rcve data bits
350 COM%=8:DAT%=RTS%:GOSUB 900                 'set stop bits
360 COM%=9:DAT%=RTP%:GOSUB 900                 'no parity
370 COM%=&HA:DAT%=1: GOSUB 900                 'enable xon/xoff xmit
380 COM%=3:GOSUB 900                           'now set Serial Driver
390 DEV%=&H35:COM%=6:DAT%=1:GOSUB 900          'select serial printer
400 LPRINT "Serial printer test: "
410 LPRINT
420 LPRINT "xmit baud rate " BAUD(TXB%)
430 LPRINT "rcve baud rate " BAUD(RXB%)
440 LPRINT "xmit data bits "  TXD%
450 LPRINT "rcve data bits "  RXD%
460 LPRINT "stop bits" STOB(RTS%)
470 LPRINT "parity" PTY$(RTP%+1)
490 END

900 DEF SEG=&H60:IO=0:RET%=0:CALL IO(DEV%,COM%,DAT%,RET%):
    RETURN
```

Reference to the Control Device chapter will provide details of the calls used in the program.

It is important to note the order of the call in statement 380. All the previous calls simply set up the configuration table which is detailed later in this chapter. Statement 380 actually re-programs the Serial Driver and the SIO.

Printer output is assumed to be directed by default to the Parallel Device driver. Statement 390 switches output to the Serial Device driver and updates the Configuration table.

All printer output within the program is then directed to the Serial Device. Provided that no other commands are given this condition will remain on exiting the program and BASIC. This means that all other output from applications through MS-DOS will also be directed to the Serial port.

This is easily demonstrated by use of the CTRL P command in MS-DOS. i.e.

A>CTRL P

A>DIR

will direct all screen output to the currently selected printer device.

A>CTRL N will cancel the CTRL P

It is important to note that the changes to the Configuration table in RAM will remain active until a cold BOOT is invoked. Applications which require to re-instate the Driver to it's former condition should temporarily store the Serial Driver configuration and then restore it as necessary.

Further, the XON/XOFF codes as defined in the Configuration table may not be compatible with the external device. If this is so then these may be altered within the configuration table. The procedure to achieve both of the above points is detailed in the section below on the Configuration table.

Statement 370 enables the XON/XOFF control of the transmit routines within the driver, i.e. receipt of XON/XOFF characters from the external device will restart/suspend the transmission procedure.

Generic differences

The Apricot generic Serial Driver provides the same support on the F1 and the Portable with the exception of a limitation in the F1 transmission rates.

The F1 does not support split rates of transmit/receive. Care must be taken in ensuring that the transmit/receive baud rate are specified equal.

In addition the F1 does not support 3600, 7200 and 19200 baud. Any attempt to specify one of these rates results in the next lower rate being used, i.e. 2400, 4800 and 9600 respectively.

The Hardware I/O addresses and port structures are different in the F1 and Portable, however this is catered for automatically by using the Control Device.

A Control Device call provides the application with full details of the Hardware configuration.

This is detailed in the next section.

User Interrupts

The Serial Device driver can be switched between:

BIOS control, and

User interrupt control (Int F0 hex).

The use of User interrupt control is limited to applications capable of accessing the 8086 and Z80 SIO registers. Details given below therefore relate only to machine code level. (For low level details of the Z80 SIO, refer to the Serial Interface chapter in the Hardware section).

The selection of User Interrupt control is achieved by the Control Device call 1A hex "Set/Reset external SIO control".

This call returns the SIO I/O addresses and structure in the PX parameter (AX register).

AL is the base I/O port address, which relates to the following offsets:

[base + 0] = Channel A data

[base + 2] = Channel A status

[base + 4] = Channel B data

[base + 6] = Channel B status

AH defines the RS-232 channnel:

0 = Channel A
1 = Channel B

The hardware chapters provide a detailed description of the SIO registers and how to access them.

Subsequent interrupts on the RS-232 channel are filtered and the interrupt F0 hex is invoked with the AX register containing "000x" which is a vector to one of the 8 conditions as given below.

For the F1 the possible settings are:

0 = Ch B TX buffer empty
2 = Ch B External Status int
4 = Ch B RX ready
6 = Ch B Special receive

For the Portable the possible settings are:

8 = Ch A TX buffer empty
10 = Ch A External Status int
12 = Ch A RX ready
14 = Ch A Special receive

The user interrupt routine handles the call as required. The nature of the interrupt is given below:

TX buffer empty - SIO ready for next character

External/Status interrupt - Line interrupt CTS/DSR etc

RX ready - SIO has received a character

Special Receive - Error in device receive channel.

Configuration table

The block entry for the Serial Device driver begins at location 0030 hex within the configuration table and is of the following format:

Offset	Function	Comment
0000	TX baud rate.	Range 1 - 15 dec. Note 1.
0001	RX baud rate.	Range 1 - 15 dec. Note 1.
0002	TX data bits.	Range 5 - 8.
0003	RX data bits.	Range 5 - 8
0004	Stop bits.	1=1, 2=1.5, 3=2.
0005	Parity check.	0 = no check, 1 = check. Note 2.
0006	Parity type.	0 = none, 1 = odd, 2 = even.
0007	TX XON/XOFF protocol.	0 = off/1 = on.
0008	RX XON/XOFF protocol.	0 = off/1 = on.
0009	XON character code.	Default DC1 (11 hex).
000A	XOFF character code.	Default DC3 (13 hex).
000B	XON/XOFF RX buffer limit.	Note 3.
000D	DTR/DSR protocol.	0 = off/1 = on.
000E	CTS/RTS protocol.	0 = off/1 = on.
000F	Number of nulls after CR.	
0010	Number of nulls x 10 after FF.	
0011	Auto LF after CR.	0 = off/1 = on.
0012	reserved.	

Notes:

1. Refer to Control Device.
2. The Parity check entry is not used.
3. The number of bytes before end of RX buffer. When the buffer fills to this point then an XOFF is transmitted. When it drops below this point then XON is transmitted.

The entries are modified from time to time by calls to the Control Device as can be seen in the example above.

If the application requires this table to be switched back to its original BOOT state following calls to the Control Device, it is essential that a copy is taken before modification. This is best achieved by making dummy calls to the Control Device GET/SET functions which return the current settings.

The Configuration table may be accessed freely but should only be altered with the generic Control device calls. One exception is the modification of the XON/XOFF characters. In certain cases the external device may send different character codes to represent XON/XOFF for example. The Application program must cater for these instances. An example of how to locate the configuration table and modify this entry is given below.

Example: Changing the XON XOFF characters

```
100 DEF SEG=&H70                    'segment for Config pointer
120 CS=PEEK(2)+(256*PEEK(3))        'Config segment
130 CO=PEEK(0)+(256*PEEK(1))        ' offset
140 DEF SEG=CS                      'define segment
150 XON=PEEK(CO+&H39)               'pick up default XON
160 XOFF=PEEK(CO+&H3A)              'XOFF
170 POKE CO+&H39,&H1                'new XON=1
180 POKE CO+&H3A,&H3                'new XOFF=3
```

The example picks up the pointer to the Configuration Table from the pointer area in RAM and saves the existing values. (Statements 120 -160). A new character for both XON and XOFF is then placed in the table. (Statements 170 and 180).

The Serial Driver, if invoked for XON/XOFF protocol, will subsequently use these two characters to filter off the devices XON/XOFF characters.

Parallel I/O Driver 3.6

Overview

The Parallel I/O Driver provides support for the Centronics output port.

Applications Interest

Facilities are provided through the Control Device for MS-DOS and Application use.

Depending upon the printer in use the Driver provides facilities to support detection of 3 status lines, they are:

FAULT SELECT PAPER OUT

Error conditions are returned without specific reference to the fault — refer to Control Device.

After Booting the system printer output is directed to the default device (either Parallel or Serial) as specified in the configuration data in the Label Sector.

A switch is available within Control Device calls to toggle between Parallel and Serial.

A 2K byte buffer is used by the Parallel Driver. If printer output is switched to Serial then the Parallel buffer is appended to the 512 byte Serial Auxiliary transmit buffer.

The Control Device also provides a call for Auto LF after CR setting.

Systems Interest

The following data is held at offset 50 hex in the configuration table:

Offset	Function
0000	auto LF after CR (0 = off, 1 = on)
0001	select line support (0 = off, 1 = on)
0002	paper empty support (0 = off, 1 = on)
0003	fault line support (0 = off, 1 = on)
0004	BIOS error report (0 = off, 1 = on)
0005	11 spare bytes

Clock driver 3.7

Overview

The Clock Driver provides the following facilities:

An internal date/time clock

A scheduler for use by peripheral drivers

A User interrupt (FF hex)

The BIOS support via the Control Device consists of three calls, they are:

Initialise, which resets the date/time clock to 00:00:00 on the 1st January 1980.

Read the date/time

Set the date/time.

The User interrupt FF hex provides Applications with a timing facility if required.

Applications interest

The Clock Driver generates a Clock Interrupt (User interrupt FF hex) every 20ms.

Applications which require a timing facility must provide an interrupt routine and modify the Software interrupt vectors to point to it.

Refer to the Guide to the BIOS chapter, section Software interrupts, for details of how to install a User interrupt routine.

In addition the following rules govern the use of the Clock interrupt:

The routine must not last more than 10ms.

The User routine must ensure that the SS, DS and ES registers are preserved.

The routine must not change the Stack unless it disables interrupts.

Calls to either the BIOS or MS-DOS are not allowed.

Systems interest

The Clock Driver maintains a System Date/Time clock relative to midnight on the 1st January 1980 to an accuracy of 2 hundredths of a second.

In addition it provides certain internal facilities for peripherals. These are:

Pre-Boot arrow flash and countdown

Sound driver timeout

Floppy and Winchester head select and movement timing

Printer character output timeout

On Booting the Apricot, a timeout is performed for 10 secs to enable the operator to change the setting of the keyboards internal battery powered clock.

At the end of the timeout automatic Booting takes place if a Bootable disk is present. In this case the System clock is not updated and reflects the time from the initialise default as stated above.

When the TIME/DATE key is pressed the System Clock is always updated from the Keyboards clock.

If the machine is still in the pre-Boot stage then an attempt to Boot then takes place.

The internal keyboard clock may be changed at any time by pressing the "Set Time" button and then entering a new time and date as prompted on the bottom line of the screen.

Note: The time and date is input without any separator characters. The Clock Driver is not updated unless the "TIME/DATE" key is pressed.

The Clock interrupt handler performs all functions necessary to maintain the system clock and all the calls necessary for peripheral timer routines. It then invokes User interrupt FF hex before returning.

The Configuration table does not hold any data related to the Clock driver.

Sound Driver 3.8

Overview

The Sound Generator hardware within the Apricot family is used by the Sound driver to produce the following:

Device

Key click

Bell

Volume, Frequency and Period setting

Calls are provided through the Control Device for these facilities.

Applications Interest

In order to produce more complex Sound features such as music, the hardware must be accessed directly by the Application.

The Hardware section describes in detail how to access the Sound Generator directly.

Systems Interest

In conjunction with the Keyboard driver, the Sound driver produces a Key Click whenever a key is detected as having been depressed. This is important from the point of view of using infra-red keyboards. If the signal is not detected when a key is pressed, a Key Click is not generated.

It thus acts as an indication to the user of correct/incorrect positioning of the Keyboard relative to the Systems Unit.

Disk Driver 3.9

Contents

Overview

The Disk Driver provides all functions for low level access of the Disks required by MS-DOS.

The Control Device calls to both Floppy and Winchester are designed to provide support at Operating System level only, i.e. for implementation of MS-DOS and other Operating Systems (OS).

Under normal circumstances all other Applications should be limited to use of the OS calls themselves and should NOT access the Control Device directly, unless otherwise stated in this chapter.

There are some exceptional cases. For example, the detection of "Disk swapped" is of importance to many Applications. Here the facilities are freely available within the Control Device. However they must be used with care. Refer to the section Disk swapping and to the Control Device chapter for the exact course of action to take.

The Disk Driver and Control Device support a number of different drive types and mixed combinations of Floppy and Winchester. Some drives however are hardware options. For full details, refer to the Hardware section.

The format of Disks is considered at two levels, i.e. Hardware physical format and Software MS-DOS format. Software "hooks" are provided within the BIOS to format a Floppy. Utilities are provided with the system software to perform this function for MS-DOS. Applications which require specific formatting should refer to the Hardware section.

The MS-DOS format is provided for completeness only. It depicts the actual layout of reserved areas on the disks, especially BOOT disks. The 'actual' format of any one Floppy or Winchester is defined in the Label Sector. Reference to the Guide to the BIOS chapter together with the appropriate MS-DOS manuals will provide full details.

Applications interest

Non MS-DOS systems

These applications are considered to be the implementation of other Operating Systems, for example CP/M.

The Control Device provides the facilities required to:

Detect the drive type

Format Floppy

Read/Write/Verify/Read + Verify sector(s)

Return disk status

Set/detect disk swapped

Return the BPB (BIOS Parameter Block) of a specified disk

The ROM Boot routine expects certain data within the Label Sector and minimum data in the BPB (refer to Label Sector section in this chapter and the Guide to the BIOS chapter).

Further items should be taken into account when preparing a Bootable disk:

The default printer device is selected from Label Sector offset 83 hex as the Parallel or Serial port.

The screen line (i.e. 200 or 256) and column (i.e. 40 or 80) mode is determined from the Label Sector offsets A0 and A1 hex respectively.

The Serial port is pre-defined in Label Sector offset B0 to CF hex.

Other sections of the Label Sector may be required to be preset but are not vital to the Boot procedure and will be OS dependant.

Drive types

The following drives are supported by the BIOS and Control Device:

Floppy : 70 Track SS
80 Track SS (not currently used)
80 Track DS

Winchester : 5 Megabyte
10 Megabyte
20 Megabyte (not currently used)

Up to 4 disks may be connected at any one time to the Apricot. Combinations of Floppy and Winchester are supported up to a maximum of 2 drives each.

Refer to the Hardware section for details of available options.

As mentioned in the MS-DOS section later the layout of a disk varies according to the Cluster (or allocation units) size. A Cluster is the smallest unit of the disk which is allocated to any file or reserved area on the disk.

The Cluster size is designed to use the disk as efficiently as possible and in general becomes larger as the capacity of the disk increases.

Label Sector

The Label Sector is the first physical sector on the Disk, i.e. Track 0 Sector 0. It has a length of 512 bytes.

The format of the complete sector is described in the chapter Guide to the BIOS.

The first 80H bytes of the Label Sector contain data relevant to OS Booting and Disk Configuration in the form of a BPB (BIOS Parameter Block). It provides the BIOS with a description of each disk (whether a Bootable Disk or not).

The BPB for each loaded disk is held in an array table in the ROM specified RAM. The Pointer area maintains a pointer to the Floppy and Winchester BPB array tables.

The BPB has the following format:

```
** WORD sector size in bytes
*  BYTE  sectors per allocation unit
   WORD number of reserved sectors
   BYTE  number of FAT's
   WORD number of directory entries
** WORD total number of sectors
   BYTE  Media descriptor byte: FCH = 70 SS
                                FDH = 80 SS
                                FEH = 80 DS
                                F8H = 5 megabyte
                                F9H = 10 megabyte
                                FAH = 20 megabyte
   WORD sectors per FAT
** BYTE  disk type:            0 = 70 track SS
                                1 = 80 track SS
                                2 = 80 track DS
                                3 = 5M winchester
                                4 = 10M winchester
                                5 = 20M winchester
   WORD Reserved
```

Notes:

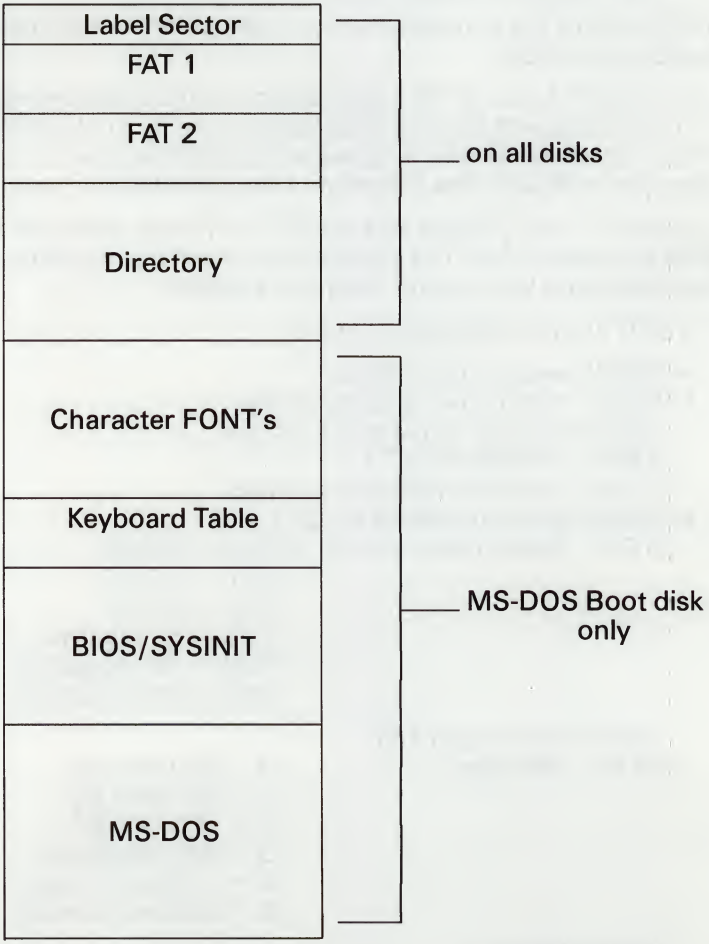
** These parameters are mandatory for the BOOT routines.

* This parameter must be non-zero.

All other parameters are OS specific, in this case MS-DOS.

MS-DOS format

The reserved part of each BOOT disk is as follows:



Only one 512 byte sector on the disk is specifically reserved and that is the first physical sector - the Label Sector.

All other information above is expected to be on the MS-DOS BOOT disks.

The location of each of the above is determined from data within the Label Sector itself.

The size of each section varies according to the Release of Software and to the cluster (or allocation) unit size of the disk.

MS-DOS 2.11 is about 17K whereas MS-DOS 3.0 is about 28K bytes. The cluster size for Winchester is bigger than for Floppy etc.

For example, the Keyboard Table is 1K on the Floppy Boot disks and 4K on the Winchester 5 megabyte. Only 1K is loaded into the Memory (and therefore the Map given in the Guide to the BIOS is always correct.)

Preparation of a Boot disk for other Operating Systems involves setting up the Label Sector exactly as required by the ROM BIOS Bootstrap routines.

Disk Formats

The Control Device provides facilities for formatting Floppy disks but there is no support for Winchester.

Formatting of a Winchester must be handled by Application or System Software alone. The BIOS does however reference a flag in the configuration table which enables or disables a Winchester park flag. (Refer to the Guide to the BIOS chapter for details - Label sector offset EE hex).

The BIOS checks the Winchester for activity every 5 secs and if there has been no activity then the driver moves the heads to the last data track.

This is to protect the data tracks at the beginning of the disk if there is an inadvertant fault. This flag ensures that the BIOS does not park the heads during the format procedure.

For details of the actual physical format refer to the relevant Hardware section.

Disk Swapping

The Disk driver has a background "Demon" which monitors the condition of the drives every 2 seconds to see whether the disk is present.

The "Demon" assumes that a disk cannot be swapped faster than this.

A number of calls are provided in the Control Device to support disk swapping. If used, however, care must be taken to ensure that MS-DOS or applicable OS are aware of the present state.

Use of the Control Device call 0009 hex (check if disk in drive x swapped) results in the BIOS flag being reset. An application making this call should therefore restore the flag to its setting if necessary by immediately calling Call 0006H.

If MS-DOS detects a disk as swapped then all internal buffers etc related to it are flushed.

A further call is available in the Control Device to obtain the BPB of a disk in a specific drive.

Systems software should invoke this call following a disk swap to obtain correct disk information.

Systems interest

Configuration data

The configuration data for disks is located in two sections of the Label Sector.

The first 80 hex bytes of a Label Sector provide BOOT detail (where applicable) together with a BIOS Parameter Block (BPB). It is important to note that the BPB also contains information vital to the BOOT procedure - refer to Label Sector section.

Further data is to be found in the Label Sector, which is related to the Winchester only, the offsets and data are as follows:

Offset	Field	Bytes	Use
00E0		14	spare
00EE	CNF__wini__park	1	parking enable flag (0 = on, nz = off)
00EF	CNF__wini__form	1	format (0 = off, nz = on)
0100	WINbad__sect	64	Up to 32 words giving logical sector numbers of bad blocks on the disk
0140	WINvol__bpb1	16	Extended BPB for volume 1
0150	WINvol__bpb2	16	" " " vol 2
0160	WINvol__bpb3	16	" " " vol 3
0170	WINvol__bpb4	16	" " " vol 4
0180	WINvol__bpb5	16	" " " vol 5
0190	WINvol__bpb6	16	" " " vol 6
01A0	WINvol__bpb7	16	" " " vol 7
01B0	WINvol__bpb8	16	" " " vol 8

A non-dedicated area of label sector starting at offset 100H is used for Winchester disk bad block tables and reserved for Multi-volume BPB's.

Configuration data for each Floppy and Winchester known to the system is loaded into RAM and may be referenced by the Pointer table in 'ROM specified RAM', as follows:

Address	Length (bytes)	Set by	Use
408H	2	BOOT	Drive booted from: 0000H = Floppy 0 0001H = Floppy 1 0002H = Winchester 1 0003H = Winchester 2
40AH	4	BOOT	Pointer to loaded boot disk header sector
40EH	4	BOOT	Reserved
412H	2	BOOT	Reserved
414H	1	BOOT	Winchester type: 0 = No Winchester 3 = 5 Megabyte R0351 Winchester 4 = 10 Megabyte R0352 Winchester 5 = 20 Megabyte Winchester
415H	1	BOOT	Floppy type: 0 = 70 track SS 1 = 80 track SS 2 = 80 track DS
416H	1	BOOT	Number of Floppy Drives
417H	1	BOOT	Number of Winchester Drives
418H	4	BOOT	Pointer to Floppy BPB array Table
41CH	4	BOOT	Pointer to Winchester BPB array Table

GSX - GIOS Details

3.10

Contents

Overview

All graphics devices are inherently different. Displays, plotters and printers are all capable of drawing lines, filling in areas and producing various forms of text. They often achieve each function in a unique way, which implies that the software controlling these devices is never compatible.

Graphics System Extension (GSX) is a support package which provides Applications with a standard interface to graphics devices.

The GSX package consists of two major components, they are:

GDOS - Graphics Device Operating System from Digital Research

GIOS - Graphics Input Output System

The GIOS is a device driver which is loaded by GSX to perform the unique features of the individual screens. A GIOS is required for each different screen resolution used in the system.

GSX loads one GIOS at a time, commencing with the default GIOS (refer to Systems interest). If the Application requests, another available device driver can be loaded instead.

GDOS and the Application communicate with a set of commands which relate to a Normalised coordinate space in the range 0 - 32767 on an X and Y axis. GDOS scales the coordinates according to information provided by the resident GIOS device driver when it is installed.

For details on how to install and use Graphics System Extension refer to Digital Research's GSX-86 Programmer's manual.

The Applications interest section of this chapter details how to call GSX functions. A number of additional functions have been included in the GIOS device drivers for the GSX-86 Release 1.3 on all Apricots. These implement, as closely as possible, new functions for GSX-86 Release 2.0 together with some ACT specific functions.

The Systems interest section details the various options implemented within the GIOS and how to install them.

Application interest

Display features

For a complete description of the Application interface to GSX, you should refer to the GSX-86 Programmers manual.

The implementation of the GIOS for the Apricot Portable and F1 Displays has the following features:

Line attributes: 7 line styles:

1 - Solid
2 - Long dash
3 - Dot
4 - Dash dot
5 - Dash
6 - Dash dot dot
7 - User defined

1 line width (1 Pixel)

Marker attributes: 5 markers (1 to 5, i.e . + * · ox)

Size: 1 - n (where n = vertical resolution i.e. 200/256 scan lines)

Text attributes: 2 character fonts:

BLOCK the current Master font as selected in the pointer table. The height is fixed according to the font, i.e. 8 or 10 pixels.

STROKE a line drawing font based upon coordinate points. The height is limited by the number of Display lines.

Rotation in multiples of 1 degree. (for STROKE only).

Fill attributes: 4 interior styles

0 - Hollow
1 - Solid
2 - Pattern
3 - Hatch

General Drawing Primitives: 4 types:

Arc
Pie Slice
Circle
Bar

All radius specifications assume an extent (distance) in the x-axis.

Colour: Portable: 640 x 200 LCD monochrome
640 x 200/256 4 colour
640 x 200/256 8 colour
640 x 200/256 16 colour

F1 : 640 x 200/256 4 colour
320 x 200/256 16 colour

The default colours are as follows:

0 Black
1 White
2 Red
3 Green
4 Blue
5 Cyan
6 Yellow
7 Magenta
8 Grey
9 Light Blue
10 Light Green
11 Light Cyan
12 Light Red
13 Light Magenta
14 Brown
15 Low-intensity White

Input devices

Input Locator: the input Locator is either:

1. Mouse, or
2. Keyboard cursor position keys

Input Valuator: the Valuator maintains a continuous count of the depression of the '+' and '—' keys. The increment is 1 unless a numeric key is pressed - in the event the increment becomes the value of the key pressed. Any non-numeric key will terminate input.

Input Choice: a value in the range 0 to 99 may be entered. If a single digit is entered then terminate input with any non-numeric key.

Input String: a string of characters terminated by carriage return.

Escapes: all Escape Op codes are implemented.

Cell arrays: Cell array display is implemented.

Cell array inquiry is NOT implemented.

Calling GSX

A detailed description of the commands, parameters and method of calling the GDOS is given in the GSX-86 Programmer's manual.

The actual Call procedure is summarised below for Application interest:

There is only one Call to the GDOS via interrupt E0 hex with the following register settings:

CX - 0473H Function code

DS:DX - Segment:Offset pointer to a parameter block which contains the following five double word Segment:Offset pointers:-

PB control array
PB + 4 input parameter array
PB + 8 input points array
PB + 12 output parameter array
PB + 16 output points array

The next section describes the specific changes for features implemented to conform to GSX-86 Release 1.3 and some additional ACT specific features.

The notation used in the GSX-86 Programmers manual is adopted in the calling sequences outlined in the next section, i.e.

contrl(x) - the control array of x integer elements

intin(x) - the input parameter array

ptsin(x) - the input coordinate points array

intout(x) - the output parameter array

ptsout(x) - the output coordinate points array

Additions to GSX 1.3

A number of additional functions are included within the GSX Release 1.3 for all Apricots.

These are mainly to conform with GSX-86 Release 2.0 specification although a few are specifically ACT functions.

Note: These additions will not necessarily be supported by ACT in future releases and are included in this manual for completeness only.

1. Set user defined line style
2. Bit block move (ACT only)
3. Set fill interior style
4. Set fill perimeter visibility
5. Exchange fill pattern
6. Fill rectangle
7. Exchange mouse form
8. Show cursor
9. Hide cursor
10. Sample mouse button states
11. Prestel operations (ACT only)

These functions together with their specific calling procedures are detailed in the remainder of this section.

Set user defined line style

This function sets the current user-defined line style pattern word in the device driver to the value in the specified pattern word, 16 bits.

The most significant bit (MSB) of the pattern word is the left most pixel displayed. This line style is used for subsequent polyline operations when the application selects user-defined line style, index 7.

Input

Contrl(1) - Escape opcode = 5
Contrl(2) - Number of input vertices = 0
Contrl(6) - Opcode = 52

intin(1) - User line style

Output

Contrl(3) - 0

Bit Block Move (ACT specific routine)

A facility has been provided for moving bit aligned blocks of data between the display and memory, using the standard GSX interface.

The routine is an escape operator with function code of 51.

Input

contrl(1) - Escape opcode = 5
contrl(2) - Number of input vertices = 10
contrl(4) - 4
contrl(6) - Function code = 51
contrl(7) - Offset of dest. bitmap
contrl(8) - Seg. of dest. bitmap
contrl(9) - Offset of srce. bitmap
contrl(10) - Seg. of srce. bitmap
contrl(11) - Offset of pattern
contrl(12) - Seg. of pattern
contrl(13) - Raster operation

intin(1) - Dest. bitmap type
intin(2) - Srce. bitmap type

ptsin(1) - Dest. origin x
ptsin(2) - Dest. origin y
ptsin(3) - Srce. origin x
ptsin(4) - Srce. origin y
ptsin(5) - Extent in x direction
ptsin(6) - Extent in y direction
ptsin(7) - Dest. bitmap width in NDC units
ptsin(8) - Dest. bitmap height in NDC units
ptsin(9) - Srce. bitmap width in NDC units
ptsin(10) - Srce. bitmap height in NDC units

Output

contrl(3) - 0

intout(1) - 0 : No errors detected
1 : Illegal source description
2 : Illegal destination description
3 : Illegal rasterop requested
4 : Illegal extent in x direction
5 : Illegal extent in y direction

The following assumptions are made:

All clipping is done by the application.

Source and destination rectangles are the same size.

Patterns are specified as 8 adjacent bytes with bits organised with the most significant bit displayed first.

If an address parameter is not required nothing need be passed. An address need not be supplied for the display.

All parameters are left unchanged.

Note: The source and destination rectangles may overlap.

Set Fill Interior Style

This function sets the fill interior style used in subsequent polygon fill operations. If the application requests an unavailable style, the area is hollow filled. GSX returns the selected style to the application. Hollow style fills the interior with the current background colour, index 0. Solid style fills the area with the currently selected fill colour.

Input

contrl(0) Opcode = 23
contrl(1) Number of input vertices = 0
contrl(3) Length of intin array = 1
contrl(6) Device handle

intin(0) Request fill interior style

0 - Hollow
1 - Solid
2 - Pattern
3 - Hatch
4 - User defined style

Output

contrl(2) Number of output vertices = 0
contrl(4) Length of intout = 1

intout(0) Fill interior style selected

Set Fill Perimeter Visibility

This function turns on or off the outline of a fill area. When visible, (the default at Open Workstation for this function), the border of a fill area is drawn in the current fill area colour with a solid line. When invisible, no outline is drawn. Any nonzero value of the visibility flag causes the perimeter to be drawn.

Input

contrl(0) Opcode = 104
contrl(1) Number of input vertices = 0
contrl(3) length of intin array = 1
contrl(6) Device handle

intin(0) Visibilty flag

zero	invisible
nonzero	visible

Output

contrl(2) Number of output vertices = 0
contrl(4) Length of intout array = 1

intout(0) Visibility selected

Exchange Fill Pattern

This function redefines the user-definable fill pattern and returns the old user-definable fill pattern definition. The inputs to this routine are two long pointers. The first points to a Fill Pattern Definition Block (FPDB), which defines the new fill pattern. The second points to a FPDB that will be overwritten with the old fill pattern. If the second pointer is zero, the old FPDB is not returned.

The format of a FPDB is shown below.

Reserved for future use, must be 1
Reserved for future use, must be 1
Reserved for future use, must be 0
16 bits by 16 bits of pattern data

For the pattern data, bit 15 of word 1 is the upper left hand bit of the pattern. Bit 0 of word 16 is the lower right hand bit of the pattern. Bit 0 is the least significant bit of the word. A bit value of 1 indicates foreground colour and a bit value of 0 the background colour. The colour used for the foreground is determined by the current fill area colour index.

The defined pattern is referenced by the Set Fill Interior Style function as style 4 and in the Fill Rectangle function.

Input

contrl(0)	Opcode = 112
contrl(1)	Number of input vertices = 0
contrl(3)	Length of intin array = 0
contrl(6)	Device handle
contrl(7 - 8)	Double word address of the user defined fill pattern
contrl(9 - 10)	Double word address of the area to copy the old fill pattern

Output

contrl(2)	Number of output vertices = 0
contrl(4)	Length of intout array = 0

Fill Rectangle

This function fills a rectangular area with the pattern defined with the Set Fill Pattern function. The rectangle is filled using the writing mode specified in intin(0).

Input	
contrl(0)	Opcode = 114
contrl(1)	Number of input vertices = 2
contrl(3)	Length of intin array = 1
contrl(6)	Device handle
contrl(7-8)	Double word address of the destinations memory form definition block (Not implemented, physical display assumed)
intin(0)	Writing mode
ptsin(0)	X-coordinate of upper left of destination rectangle in RC/NDC
ptsin(1)	Y-coordinate of upper left of destination rectangle in RC/NDC
ptsin(2)	X-coordinate of lower right of destination rectangle in RC/NDC
ptsin(3)	Y-coordinate of lower right of destination rectangle in RC/NDC
Output	
contrl(2)	Number of output vertices = 0
contrl(4)	Length of intout array = 0

Exchange Mouse Form

This function redefines the cursor pattern displayed during locator input or any time the cursor is shown (see Show Cursor function), and returns the old cursor definition. The inputs to this routine are two long paramaters. The first points to a cursor definition block which is to be used as the new cursor. The second points to a cursor definition block which will be overwritten with the old cursor definition block. If the second pointer is zero, the old cursor definition block is not returned.

Format of Cursor Definition Block

X-coordinate of hot spot
Y-coordinate of hot spot
Reserved, must be 1
Colour Index
Reserved, must be 0
16 by 16 bit of cursor mask
16 by 16 bit of cursor data

For the cursor mask and data, bit 15 of word 1 is the upper left hand bit of the pattern and bit 0 of word 16 is the lower right bit of the pattern. Bit 0 is the least significant bit (LSB) of the word. A bit value of 1 indicates foreground colour and a bit value of 0 indicates background colour. The colour used for foreground colour is determined by the colour index.

The hotspot is the location of the pixel (relative to the upper left corner of the cursor) that lies over the pixel whose address is returned by the input locator function.

The cursor is drawn as follows:

The data under the cursor is saved so it can be restored when the cursor moves.

The mask is ANDed with the data on the screen.

The cursor data is XORed with the result of the previous step and displayed on the screen.

Input	
contrl(0)	Opcode = 111
contrl(1)	Number of input vertices = 0
contrl(3)	Length of intin array = 0
contrl(6)	Device handle
contrl(7 - 8)	Double word address of the new cursor definition block
contrl(9 - 10)	Double word address to receive the old cursor definition block
Output	
contrl(2)	Number of output vertices = 0
contrl(4)	Length of intout array = 0

Show Cursor

This function displays the current cursor. The cursor moves on the display surface based on information input from a mouse or the cursor control keys.

The Show Cursor function and the Hide Cursor function are closely related. Once the cursor is visible, a single Hide Cursor causes the cursor to disappear. GSX maintains the number of times the Hide Cursor function is called. The Show Cursor function must be called the same number of times for the cursor to reappear. For example, if the Hide Cursor function is called four times, the Show Cursor function must be called four times for the cursor to reappear.

However, the Show Cursor function provides a reset flag in intin(0). If intin(0) is zero, the cursor appears on the screen reguardless of the number of Hide Cursor calls. A nonzero value for intin(0) affects the Show Cursor function as described in the previous paragraph.

Input

contrl(0) Opcode = 122
contrl(1) Number of input vertices = 0
contrl(3) Length of intin array = 1
contrl(6) Device handle

intin(0) Reset flag

0 = ignore number of hide cursor calls
nonzero = normal show cursor behavior

Output

contrl(2) Number of output vertices = 0
contrl(4) Length of intout array

Hide Cursor

This function removes the cursor from the display surface. This state is the default condition set at Open Workstation. The cursor can appear in a new position when the aplication calls the Show Cursor function because GSX updates the position based on information input from a mouse or the cursor control keys.

Refer to the Show Cursor function for a description of how the number of Show Cursor calls affects the Hide Cursor function.

Input

contrl(0) Opcode = 123
contrl(1) Number of input vertices = 0
contrl(3) Length of intin array = 0
contrl(6) Device handle

Output

contrl(2) Number of output vertices = 0
contrl(4) Length of intout array = 0

Sample Mouse Button State

This function returns the current state of the mouse buttons. The leftmost mouse button is returned in the least significant bit of the word. A bit value of 1 indicates the key is currently depressed, a bit value of 0 indicates the key is up.

This function also returns the current x, y position of the cursor.

Input

contrl(0) Opcode = 124
contrl(1) Number of input vertices = 0
contrl(3) Length of intin array = 0
contrl(6) Device handle

Output

contrl(2) Number of output vertices = 1
contrl(4) Length of intout array = 1

intout(0) Mouse button state

ptsout(0) X position of cursor in NDC/RC units
ptsout(1) Y position of cursor in NDC/RC units

Prestel Operations. (ACT specific routine)

An escape op is provided to allow the setting of attributes for PRESTEL text. The only attribute initially available is double height. Colour is handled using standard GSX text functions.

Input

Contrl(1) Escape opcode = 5
Contrl(2) Number of input vertices = 0
Contrl(6) Op code = 55

intin(1) Set text single height = 0
Set text double height = 1

Output

Contrl(3) 0

An escape operation is also provided to allow the writing of PRESTEL text. This operation depends on the standard font pointer refering to a double width font for 40 column mode PRESTEL.

Input	
contrl(1)	Escape opcode = 5
contrl(2)	Number of input vertices = 1
contrl(4)	Number of characters in string
contrl(6)	Opcode = 56
intin	Word character string = 0
ptsin(1)	X-coordinate for start of string
ptsin(2)	Y-coordinate for start of string
Output	
contrl(3)	0

Notes:
This routine assumes that characters require no clipping. The x,y position for the character is the bottom left corner of the character cell NOT the character origin.

The writing mode is always transparent (mode 2) so that PRESTEL can use bar for the background and then draw 'OR' the character onto this.

Systems interest

System Files

The GSX system comprises 2 fixed files

GRAPHICS.EXE
ASSIGN.SYS

together with a variable number of GIOS device driver files which are defined in the ASSIGN.SYS file in a pre-determined format as discussed in the GSX-86 Programming manual.

The contents of ASSIGN.SYS may be simply inspected by invoking an MS-DOS command as follows:

```
A>TYPE ASSIGN.SYS

06 F116        .SYS  F1 16 colour 40 column
01 PC          .SYS  Apricot PC monochrome
02 F1          .SYS  F1 4 colour 80 column
03 PORTLCD     .SYS  Portable LCD
04 PORTCOL3    .SYS  Portable 8 colour
05 PORTCOL4    .SYS  Portable 16 colour

A>
```

In order to invoke a different default device edit the ASSIGN.SYS file as described in the GSX-86 Programmer's manual. Remember that ASSIGN.SYS may contain any number of GIOS Device drivers.

The first occurrence in the file is the Default GIOS driver and it should always be the largest of all required device drivers. It is loaded by GSX which only reserves sufficient memory for the default driver.

Applications may invoke other GIOS device drivers by using a standard GSX Call 'Open workstation' which references the file by the unique number which should be assigned to it in ASSIGN.SYS. Only one GIOS may be resident at a time.

Appendices

Appendix A - Diagnostics Error Codes

When the system is booted up a check is made to see whether it is a cold or warm start. If it is a cold (power-up) start the ROM resident diagnostic program is run to check the integrity of the hardware.

If an error is found the diagnostic program is terminated and the error number is shown on the start-up screen. See Table 1.

The ROM BIOS enters the boot procedure, displays the start-up screen and eventally starts looking for a bootable disk. Boot disk errors are listed in Table 2. Note that the screen display of an error from the diagnostics is overwritten by the boot procedure.

Both types of error are indicated by a large 'X' on the screen, followed by a one or two digit error code.

Table 1. Boot ROM test error codes

Code	Test Failed
20	ROM TEST: Calculates the checksum of the ROM and compares it to that stored at the start of ROM.
22	SIO TEST: Register test of Z80 SIO
23	CRTC TEST: Register test of the 6845 video controller. This test is skipped if there is no colour RAM present.
24	DISPLAY RAM: Checks the LCD display RAM. If any colour RAM is found that is also checked.
25	SYSTEM RAM: Checks the system and any extension RAM.
27	PIC TEST: Register test of the 8259 programmable interrupt controller.
28	FDC TEST: Register test of the WD 2797 floppy disk controller.
29	PIT TEST: Register test of the clock chip.
33	CLOCK INT: Checks whether a clock interrupt can be generated
35	DRIVE TEST: Checks whether drive 0 exists and can be restored and stepped.
37	DMA TEST: Tests the operation of the 8237 DMA controller by getting it to do a memory to memory move.
38	VOICE TEST: Tests the 6301 speech processor. The result of the 6301's own internal ROM and RAM check are also obtained.

Table 2. Disk error codes

Code	Reason
2	Drive not ready or disk removed during boot.
4	CRC error, corrupt sector format.
6	Seek error; unformatted or corrupt disk, or faulty floppy drive.
7	Bad media, corrupt media block.
8	Sector not found; unformatted or corrupt disk, or bad load address in label.
11	Bad read, corrupt data field on disk.
12	Disk failure, disk hardware or media fault.
99	Non-System disk (Not a valid boot disk, or disk incompatible with drive, or invalid load address). Note: if another loadable disk is found, this message will only be displayed very briefly.

Appendix B - Default Keyboard Table

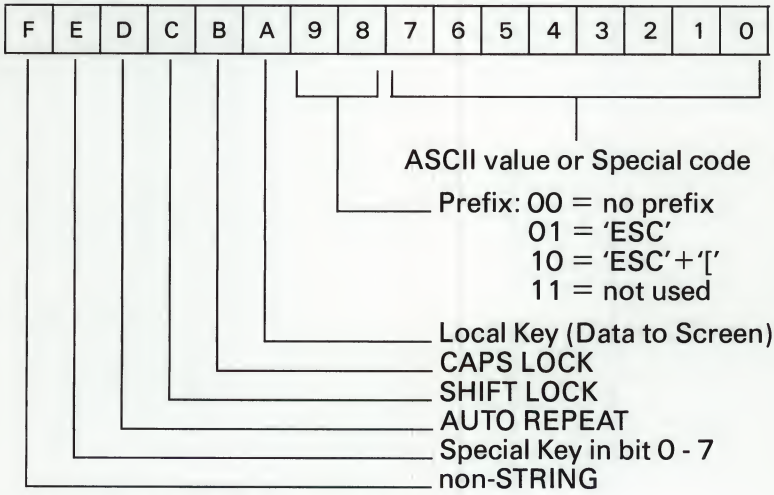
The Keyboard Table and its function is described in detail in the Keyboard Driver chapter. The purpose of this appendix is to provide a quick reference guide to the format of the ROM based keyboard table.

The default table is split into three sections each 104 words long to accomodate NORMAL, SHIFTed and CONTROL modes.

Although these 3 sections respectively follow each other in the ROM table, they are depicted below in "down-code" sequence as returned by the Keyboard Driver showing each of the three keyboard modes together.

Each entry in the table is 1 word in the following format:

(0) <— 15 bit STRING pointer —>



Attribute bits A - F are: 'ON' if set to 1
'OFF' if set to 0.

The exception to this format is when the "Non-STRING" attribute F is set 'OFF' implying that the down-code is to be translated into a string of data. In this case the format is:

bit 0 - bit E : String offset of 15 bits (i.e. 32K)
bit F : set 0 = STRING

The 15 bit address points to an entry which must be after the end of the first 312 words of the table.

The actual entry is of a different format as it is of variable length. The format is:

Byte 'n' : String byte count 'n'

Byte 'n' : Atrributes

'n' Bytes : Data

Examples of String entries are to be found in the table in the format of a pointer as follows:

<string name>

e.g. <80 Column Apricot>

Note: The 6 Membrane Keys MB1 - MB6 are not available on the Portable or the F1.

		Normal		Shift		Control	
Legend	**Down code**	**Data**	**Attribs. Normal**	**Data**	**Attribs. Shift**	**Data**	**Attribs. Control**
F1/HELP	01	B1H	10000000	B2H	10000000	<80 col Apricot Comp.> (see Note 1)	
F2/UNDO	02	B2H	10000000	B2H	10000000	<80 col colour>	
F3/REPEAT	03	B3H	10000000	'U'	10000001	<40 col Colour> (see Note 2)	
F4/CALC	04	B4H	10000000	06H	11000000	<80 col Apricot Comp.> (see Note 3)	
F6/PRINT	05	B5H	10000000	26H	10000101	07H	11000000
F7/INTR	06	B6H	10000000	B6H	10000000	B6H	10000000
F8/MENU	07	B7H	10000000	B7H	10000000	B7H	10000000
F9/FINISH	08	B8H	10000000	B8H	10000000	B8H	10000000
MB1	09	B9H	10100000	B9H	10100000	B9H	10100000
MB2	10	BAH	10100000	BAH	10100000	BAH	10100000
MB3	11	BBH	10100000	BBH	10100000	BBH	10100000
MB4	12	BCH	10100000	BCH	10100000	BCH	10100000
MB5	13	BDH	10100000	BDH	10100000	BDH	10100000
MB6	14	BEH	10100000	BEH	10100000	BEH	10100000
^	15	' '	10110000	'^'	10110000	' '	10110000
1!	16	'1'	10110000	'!'	10110000	ACH	10000000
2@	17	'2'	10110000	'@'	10110000	ABH	10000000
3#	18	'3'	10110000	'#'	10110000	00H	10000000
4#	19	'4'	10110000	'#'	10110000	9DH	10000000
5%	20	'5'	10110000	'%'	10110000	F8H	10000000
6$	21	'6'	10110000	'$'	10110000	9BH	10000000
7&	22	'7'	10110000	'&'	10110000	'''	10000000
8*	23	'8'	10110000	'*'	10110000	F1H	10000000
9(	24	'9'	10110000	'('	10110000	'~'	10110000
0)	25	'0'	10110000	')'	10110000	'|'	10110000
—_	26	'—'	10110000	'_'	10110000	'—'	10110000
=+	27	'='	10110000	'+'	10110000	'='	10110000
Backspace	28	08H	10100000	08H	10100000	08H	10100000
Percentage	29	'%'	10100000	'%'	10100000	'%'	10100000
Multiply	30	'*'	10100000	'*'	10100000	'*'	10100000
Divide	31	'/'	10100000	'/'	10100000	'/'	10100000
Subtract	32	'—'	10100000	'—'	10100000	'—'	10100000
Addition	33	'+'	10100000	'+'	10100000	'+'	10100000
TAB/BACK	34	09H	10100000	09H	10100000	09H	10100000

Notes:

1. This directs screen output to the default screen using Apricot Compatible mode. The default screen is the LCD on the Apricot Portable; the normal display monitor on the Apricot F1.
2. 40 column mode.
3. 80 column Apricot Compatible mode in any two colours. This is the same mode as defined by CTRL + F1 on the Apricot F1, but directs screen output to the colour monitor on the Portable (instead of the LCD).

		Normal		Shift		Control	
Legend	**Down code**	**Data**	**Attribs. Normal**	**Data**	**Attribs. Shift**	**Data**	**Attribs. Control**
Q	35	'q'	10111000	'Q'	10111000	11H	10111000
W	36	'w'	10111000	'W'	10111000	17H	10111000
E	37	'e'	10111000	'E'	10111000	05H	10111000
R	38	'r'	10111000	'R'	10111000	12H	10111000
T	39	't'	10111000	'T'	10111000	14H	10111000
Y	40	'y'	10111000	'Y'	10111000	19H	10111000
U	41	'u'	10111000	'U'	10111000	15H	10111000
I	42	'i'	10111000	'I'	10111000	09H	10111000
O	43	'o'	10111000	'O'	10111000	0FH	10111000
P	44	'p'	10111000	'P'	10111000	10H	10111000
[<	45	'['	10110000	'<'	10110000	'['	10110000
] >	46	']'	10110000	'>'	10110000	']'	10110000
HOME/ESC H	47	'H'	10000001	'H'	10000001	'H'	10000101
CLEAR	48	'E'	10000001	'E'	10000001	'z'	10000001
7	49	'7'	10111000	'7'	10111000	'7'	10100000
8	50	'8'	10111000	'8'	10111000	'8'	10100000
9	51	'9'	10111000	'9'	10111000	'9'	10100000
CAPS LOCK	52	01H	11000000	01H	11000000	01H	11000000
A	53	'a'	10111000	'A'	10111000	01H	10111000
S	54	'b'	10111000	'S'	10111000	13H	10111000
C	55	'd'	10111000	'D'	10111000	04H	10111000
F	56	'f'	10111000	'F'	10111000	06H	10111000
G	57	'g'	10111000	'G'	10111000	07H	10111000
H	58	'h'	10111000	'H'	10111000	08H	10111000
J	59	'j'	10111000	'J'	10111000	0AH	10111000
K	60	'k'	10111000	'K'	10111000	0BH	10111000
L	61	'l'	10111000	'L'	10111000	0CH	10111000
;:	62	';'	10110000	':'	10110000	';'	10110000
' "	63	'''	10110000	'"'	10110000	'''	10110000
RETURN	64	0DH	10100000	0DH	10100000	0DH	10100000
Line Insrt Char	65	40H	10100001	40H	10100001	40H	10100001
Line Del Char	66	7FH	10100000	7FH	10100000	7FH	10100000
4	67	'4'	10111000	'4'	10111000	'4'	10111000
5	68	'5'	10111000	'5'	10111000	'5'	10111000
6	69	'6'	10111000	'6'	10111000	'6'	10111000

Legend	Down code	Data	Attribs. Normal	Data	Attribs. Shift	Data	Attribs Control
LH shift	70	03H	11000000	03H	11111000	03H	11000000
Z	71	'z'	10111000	'Z'	10111000	1AH	10111000
X	72	'x'	10111000	'X'	10111000	18H	10111000
C	73	'c'	10111000	'C'	10111000	03H	10111000
V	74	'v'	10111000	'V'	10111000	16H	10111000
B	75	'b'	10111000	'B'	10111000	02H	10111000
N	76	'n'	10111000	'N'	10111000	0EH	10111000
M	77	'm'	10111000	'M'	10111000	0DH	10111000
, <	78	','	10110000	'<'	10110000	','	10110000
. >	79	'.'	10110000	'>'	10110000	'.'	10110000
/ ?	80	'/'	10110000	'?'	10111000	'/'	10110000
RH shift	81	02H	11000000	02H	11000000	02H	11000000
Up arrow	82	'A'	10100001	<Fast screen scroll>		<LCD contrast up>	
Scroll	83	0AH	10100000	00H	10000000	00H	10000000
1	84	'1'	10111000	'1'	10111000	'1'	10111000
2	85	'2'	10111000	'2'	10111000	'2'	10111000
3	86	'3'	10111000	'3'	10111000	'3'	10111000
ESC	87	1BH	10100000	1BH	10100000	1BH	10100000
CONTROL	88	04H	11000000	04H	11000000	04H	11111000
SPACE	89	' '	10100000	' '	10100000	' '	10100000
STOP	90	05H	11000000	05H	11000000	05H	11000000
Left arrow	91	'D'	10100001	'D'	10100001	<Bell volume dn>	
Down arrow	92	'B'	10100001	<Smooth scrolling>		<LCD contrast down>*	
R arrow	93	'C'	10100001	'C'	10100001	<Bell volume up>	
0	94	'0'	10100000	'0'	10100000	'0'	10100000
	95	'.'	10100000	'.'	10100000	'.'	10100000
ENTER	96	0DH	10100000	0DH	10100000	0DH	10100000
F5/VOICE	97	B9H	10000000	08H	11000000	08H	11000000
spare	98		11111000		11111000		11111000
spare	99		11111000		11111000		11111000
spare	100		11111000		11111000		11111000
spare	101		11111000		11111000		11111000
spare	102		11111000		11111000		11111000
spare	103		11111000		11111000		11111000
spare	104		11111000		11111000		11111000

* Apricot Portable only

String name	String Length	Attribs.	STRING		
<LCD contrast up>	3	10100100	27	'y'	'E'
<LCD contrast dn>	3	10100100	27	'x'	'E'
<Fast screen scroll>	3	10100100	27	'y'	'D'
<Smooth scrolling>	3	10100100	27	'x'	'D'
<Keyclick volume up>	3	10100100	27	'y'	'F'
<Keyclick volume dn>	3	10100100	27	'x'	'F'
<80 column Apricot>	3	10100100	27	'7'	'L'
<80 col Colour>	3	10100100	27	'7'	'C'
<40 col Colour>	3	10100100	27	'7'	'F'
<80 column Apricot>	3	10100100	27	'7'	'B'

The remainder of the 1024 bytes of the default table are zero.

Appendix C Ascii codes

The Apricot supports a default 128 character Ascii set in ROM and optionally a full 256 character Ascii set in RAM.

The full character set is tabled below in Figure 1.

Following power up the pre-Boot software uses the 128 character Font in the ROM.

The Bootstrap procedure searches for the file FONT.SYS which contains an 8 x 8 and an 8 x 10 Font to provide support for 200 and 256 scan lines respectively. If found then the file is loaded into RAM at a fixed position (Refer to Guide to the BIOS Memory Map).

A constant at offset 0A0H in the Disk Label Sector (refer to the chapter Guide to the BIOS) indicates the number of scan lines and hence the Font to be used. The Bootstrap procedure references this constant and then sets up pointers to the Master and Active Font.

The pointer table in the 'ROM specified RAM' (refer to Memory Map in chapter Guide to the BIOS) has two entries which indicate where the Master and Active Fonts are located. Each entry consists of the following:

Offset/Segment pointer to the character set

Length of the Font in bytes

The Master Font and Active Font pointers are set by the Bootstrap routine.

The Master Font pointer points to the base of the table in RAM where FONT.SYS is loaded.

If FONT.SYS is not on the disk then the Master Font and Active Font pointers both point to the ROM Font.

If FONT.SYS is on the disk, it is loaded into RAM and the Active Font pointer is set according to the parameter in the configuration data of the Label Sector. It will point to either the 8 x 8 or the 8 x 10 Font.

Normally Applications will not need to alter the pointers. However, Applications may use alternative/multiple character sets by loading them in free memory and modifying the Active pointer and length.

Figure 1. Default 256 character ASCII set

Appendix D — Circuit Diagrams

Appendix E ESCape sequence reference table

The ESCape sequence Table in the Screen Driver chapter lists the sequences in Ascii ascending order.

This appendix lists them by the following activities for quick reference:

1. Specials
2. Character attributes
3. Screen attributes
4. Colour
5. Cursor positioning
6. WP primitives
7. Driver environment
8. Keyboard
9. Generic obsoletes

Specials

CHR	HEX	M/C	ACTION
&	26	PFA	Print Page Outputs the contents of the Screen to the line printer. A form-feed is executed first.
'	27	PFA	Print line Outputs the entire line that the cursor is at to a connected line printer. No form feed is executed.
*	2A	A	Change to second character font.
8	38	PFA	Set literal/Test mode ON The escape sequence tells the screen driver to perform the following action on receiving the next character: Ignore the fact that it is a control code (<20H) and print the character associated with it. This means that the font cell characters under (20H) are printed, rather than obeyed. The ESCape must be sent for each character.
?	3F	A	Enter CALC mode This escape sequence switches on the internal BIOS calculator. It is the same as pressing the "calc" key.
[	5B	PFA	ANSI lead-in character. Refer to section ANSI ESCape sequences of the Screen driver chapter for details of the ANSI codes supported.
}	7D	PF	GSX private - this call is documented for completeness only. It is not supported.

Character attributes

CHR	HEX	M/C	ACTION
0	30	PFA	Sets underline mode. All characters printed have a single line of pixels placed under them to simulate underline.
1	31	PFA	Reset underline mode. Cancels the mode set above.
9	39	PFA	Set strikeout mode ON. All following characters are displayed with a horizontal line through them at the centre. This is widely used for deleting data within legal documents.
:	3A	PFA	Set strikeout mode OFF. The code reverts the action of "9" (39H).
p	70	PFA	Enter reverse video mode. All characters printed after this sequence are displayed in inverse video.
q	71	PFA	Cancel reverse video mode. Restores the writing mode to the state it was in before ESC 'p' was executed.

Screen attributes

CHR	HEX	M/C	ACTION
(	28	PFA	Set High intensity Mode Shadow-prints all characters to give the effect of high intensity characters.
)	29	PFA	Set Low intensity Mode Clears the mode set above.
+	2B	A	Clear all high intensity characters.
,	2C	PFA	Set Window size. Takes four parameters in Ascii: <1> Top line + 31 <2> Bottom line + 31 <3> Left-hand column + 31 <4> Right-hand column + 31
-	2D	A	Clear all low-intensity characters.
.	2E	PFA	Reset Window Size. Resets the Window size set by code 2C

Colour

CHR	HEX	M/C	ACTION
5	35	PF	Set foreground colour. This escape sequence takes one parameter which is from "0" (30H) to "?" (3FH). This gives 16 possible indexes. For a list of indexes and colours represented by them see escape sequence "]" (5DH). See also the diagram for ESC "6" (36H) below.
6	36	PF	Set block or background colour. This escape sequence sets the colour of all pixels in the character cell that do not make the actual character shape. As above, it takes one parameter. Diagram of a character cell: Background. (all 0's) 00000000 00111100 00100100 001001 — Foreground or text 00111100 colour. (all 1's) 00100100 00100100 00000000
]	5D	PF	Set palette code. This escape sequence takes two arguments. The first is the index which needs to be changed, and the second is the colour. e.g. PRINT CHR$(27) "]05" Sets index 0 (in this case the background) to colour 5. Refer to the Colour section of the Screen Driver chapter for full details of index, colour and default Palette settings.

Cursor positioning

CHR	HEX	M/C	ACTION
;	3B	PFA	Position cursor to start of status line. Position the cursor to the start of the status line, i.e. the 25th line of the Screen. The cursor remains on this line until a position cursor command is given.
A	41	PFA	Cursor UP.
B	42	PFA	Cursor DOWN.
C	43	PFA	Cursor RIGHT.
D	44	PFA	Cursor LEFT.
E	45	PFA	Clear screen. The current window is cleared, and the cursor is homed.
H	48	PFA	Home Cursor. The cursor is placed at the top left hand corner of the current text window.
Y	59	PFA	Position Cursor. This sequence takes two parameters. They are the line number and column number in normalised Ascii. e.g. PRINT CHR$(27) "Y" CHR$(10+31) CHR$(15+31) will position the cursor at line 10, column 15.
j	6A	PFA	Save cursor position The current cursor position is noted within the BIOS.
k	6B	PFA	Restore cursor The cursor is restored to the position it was in before ESC 'j' was executed.

WP Primitives

CHR	HEX	M/C	ACTION
@	40	PFA	Enter insert mode. After this escape sequence is issued, whenever a character is printed, all the characters to the right of it will be shifted right one place and the character will be inserted in the space created.
I	49	PFA	Reverse-index and line-feed. This sequence moves the cursor up one line, however if the cursor is at the top of a window then a scroll DOWN of the whole window is performed.
J	4A	PFA	Erase to end of Page. The sequence first of all erases all characters from the cursor position to the end of the current line, and then all subsequent lines below the cursor till the end of the current window or page.
K	4B	PFA	Erase to end of line. This escape sequence erases all characters from the cursor position to the end of the defined right-hand side margin.
L	4C	PFA	Insert line. This sequence places the cursor to the beginning of the current line, and then inserts one line below the current cursor position by scrolling all subsequent lines down by one place.

WP Primitives (Continued)

CHR	HEX	M/C	ACTION
M	4D	PFA	Delete line. This sequence places the cursor to the beginning of the current line and scrolls all lines under it up by one place.
N	4E	PFA	Delete character. The character under the cursor is cleared, and all characters to the right are scrolled left by one position. This is active in the defined right-hand margin space.
O	4F	PFA	Exit Insert Mode. This sequence reverses the effect of ESC "@" (40H)
P	50	PFA	Insert single character. This sequence scrolls all characters from the current cursor position to the defined right-hand margin right by one place.
Q	51	PFA	Scroll left. Takes one parameter which is the number of columns plus 31 that the screen is to be scrolled. This is only active in the current window.
R	52	PFA	Scroll right. As above, except scrolling is right rather than left.
S	53	PFA	Scroll up. As above, but scrolling is UP.
T	54	PFA	Scroll down. As above, but scrolling is down.

WP Primitives (Continued)

CHR	HEX	M/C	ACTION
b	62	PFA	Erase from start of page All characters from the top left-hand corner of the current defined display page size to the current cursor position are cleared.
h	68	PFA	Reverse tab This sequence has the opposite effect of control code 09H. It performs a tabulation operation to the left rather than the right.
l	6C	PFA	Erase line The line which the cursor is on is cleared. Note that no scrolling takes place
o	6F	PFA	Erase to start of current line. All characters from the start of the current line up to and including the character under the cursor are cleared.
v	76	PFA	Wrap at end of line. This escape sequence indicates that the normal screen driver action when the cursor reaches the end of a line should be employed. The action is to return the cursor to the beginning of the next line on the screen.
w	77	PFA	Discard at end of line. When the cursor reaches the end of the current line, it will remain there.

CHR	HEX	M/C	ACTION
7	37	PF	Set screen environment. This sequence is followed by one of the following mode parameters: "0" - Apricot PC & Xi monochrome compatibility "1" - 80 column full colour display "2" - 40 column mode "3"- Apricot PC & Xi monochrome compatibility Refer to section Screen environment in the Screen driver chapter for operational details.
F	46	PFA	Enter VT52 Graphics mode. VT52 mode displays the VT52 graphics character represented by the Ascii value of the Apricot lower case letters. Other characters are not supported.
G	47	PFA	Exit VT52 Graphics mode. Invoke this mode to exit VT52. Failure to do this results in non-lower case letters being incorrectly displayed.
Z	5A	PFA	Identify as VT52. This escape sequence is included as most of the screen driver is DEC VT52 compatible. After issuing this sequence the keyboard buffer is filled with three characters which can be read by an application to determine the device type.

Driver environment (Continued)

CHR	HEX	M/C	ACTION
'	60	PFA	Save environment The first three environment flags are saved. They can then be temporarily changed and restored by another sequence.
a	61	PFA	Restore environment Returns the first three environment flags to their state just after code 60.
x	78	PFA	Set environment flags. Takes one parameter:
		PF	G - reset screen palette
		PF	$ - set to apricot comp mode on default screen
			1 - enable line 25
			2 - nothing
			3 - nothing
			4 - nothing
			5 - cursor off
			6 - nothing
			7 - nothing
			8 - set auto LF on receipt of CR
			9 - set auto CR on receipt of LF
			A - nothing
			B - nothing
			C - nothing
			D - smooth screen scrolling
		P	E - LCD contrast up
		PF	F - bell volume up

Driver environment (Continued)

CHR	HEX	M/C	ACTION
y	79	PFA	Reset environment flags. Takes one parameter:
			1 - disable line 25 2 - nothing 3 - nothing 4 - nothing 5 - cursor on 6 - nothing 7 - nothing 8 - no auto LF on receipt of CR 9 - no auto CR on receipt of LF A - nothing B - nothing C - nothing
		PF	D - fast screen scrolling
		P	E - LCD contrast down
		PF	F - bell volume down
z	7A	PFA	Reset all screen drivers.
			Sets all screen drivers to power-on status.

Keyboard interaction

CHR	HEX	M/C	ACTION
$	24	PFA	Transmit Character Sends the character under the cursor into the keyboard buffer.

4	34	PFA	Change the representation of a key
			
			This escape sequence takes 3 paramaters. They are:
			
			<1> - Key mode (ascii) :
			1 = normal
			2 = shift
			3 = control
			<2> - Key number — 1 :
			0 = help... etc.(refer to Appendix B)
			<3> - New key character :
			Ascii char. or hex equivalent.
			
			`10 PRINT CHR$(27)+"4"+"3"`
			`+CHR$(72)+"C"`
			
			This example changes the key with the legend "C" (downcode number 72), in control mode, to generate an ascii "C" (43H) as opposed to a binary 03H.
			
			The key has the default attribute of AUTO-REPEAT. Refer to Appendix B for a list of keys, their corresponding character value, down code and attributes.
			
			Note: The Screen Driver does not accept non-printable characters (range 0 to 31) in ESCape sequences therefore no Key or Key character can be programmed with a value below 32. See <2> and <3> above.

Keyboard interaction (Continued)

CHR	HEX	M/C	ACTION
/	5C	PFA	Place key in keyboard buffer. This sequence takes one parameter which is placed in the keyboard buffer. If the buffer is full the bell will sound and the character will be ignored. eg: PRINT CHR$(27) "/R" will place an "R" into the keyboard buffer.
n	6E	PFA	Return Cursor position. This sequence places a valid ESC "Y" sequence representing the position of the cursor at the current position in the keyboard buffer. i.e 4 bytes of data: 27+"Y"+(31+column)+(31+row) where column and row have a base of 1.

Generic obsoletes

CHR	HEX	M/C	ACTION
/	2F	A	Set membrane key LED's.
<	3C	A	Display time on MSCREEN.
U	55	A	Enable dual-output to MSCREEN
V	56	A	Disable dual output.
W	57	A	Output text to MSCREEN only.
c	63	A	Disable MSCREEN scrolling.
d	64	A	Enable MSCREEN scrolling.
e	65	A	Switch MSCREEN cursor ON.
f	66	A	Switch MSCREEN cursor OFF.
g	67	A	Disable Time and Data display on MSCR5EEN.
r	72	A	MSCREEN echo enable.
s	73	A	MSCREEN echo disable.

Appendix F Language interfaces

Overview

The programming examples in this manual are provided in Interpretive Basic.

While it is recognised that the majority of Software products are written in either assembler or compiled High level languages it does not necessarily imply that these tools are universally available.

For this reason, together with the overriding consideration that all examples should be in a common language, Interpretive Basic was chosen.

This appendix is provided to support the programming examples given throughout the manual and as an introduction to interfacing in Microsoft Compiled Basic and 'C'.

Interpretive Basic and the compiled languages mentioned do not provide a compatible means of interfacing to machine code.

The most common problem being that pointers to parameter blocks are either in the incorrect format or on the stack instead of in a register.

Another problem is that, in the case of BIOS, GSX and other installed Device driver calls, pointers need to be set up containing the Applications Data Segment. The address of the Data Segment is not however accessible by many languages.

Apart from these points, languages such as Interpretive Basic have idiosyncrasies which are not at first apparent to the Application programmer with little experience in this field.

Therefore the requirement for interfaces of all kinds and care in using them is needed. This appendix does not attempt to cover all possibilities but merely to support the programming examples given and to act as an introduction to the subject.

The Control device chapter provides details of how to interface to the BIOS in assembler and BASIC. For other external software such as GSX reference to the appropriate manual will provide full programming details.

Interpretive Basic

This is one of the most easily used High Level languages because of it's interactive nature but it has not been developed with a view to interfacing to machine code within the Application bounds.

All of the reasons for this fall outside the scope of this manual however the relevant points are as follows:

1. The use of the Call statement with a string parameter results in the interpreter putting a string descriptor pointer on the stack - not the address of the string. The descriptor is a 3 byte packet containing the string length and the address of the data itself.
2. No functions are available to obtain the value of the Data Segment register.
3. The location of string variables can change during execution of the Application.

As mentioned in the overview many calls to systems software require pointers to parameter blocks.

In addition, the machine code interface must be located within a reserved data area, and this can be achieved by creating it in either a string variable or an integer array. Both may cause problems without due care.

The Data segment is not available because of historical reasons. The only solution is to use an interface or subroutine in machine code to obtain it. The implementation of such an interface is given below.

The location of a string however poses several problems. Strings are by far the most convenient way to store a machine code routine and they are also widely used as parameter blocks.

Interpretive Basic moves strings in memory as it creates them especially when using concatenation.

The following precautions should therefore be taken:

1. Always ensure that the address of a string is correct just prior to calling the machine code routine.
2. Ensure that the string area is not unneccesarily cluttered with work data by executing a FRE(" ") function whenever convenient.

   Microsoft Basic does this from time to time automatically but where concatenations are frequent it is possible that a potential string field overlaps other reserved areas. The machine code routine has no knowledge of the interpreter and may alter the string and corrupt other data.

   In either case caution should be taken to ensure that a string pointer and the contents of the string are unchanged.

The Data Segment register

The routine below creates a machine code interface which when executed returns the value of the Data Segment register in an integer variable. On entry the stack has the following entries:

SP + 4 Address of integer variable
SP + 2 Segment of Return address
SP + 0 Offset of Return address

```
9000 REM routine at GETSEG
9010 DATA &H55,&H56                          :REM push bp:push si
9020 DATA &H89,&HE5                          :REM mov bp,sp
9030 DATA &H8B,&H76,&H08                     :REM mov si,[bp+8]
9040 DATA &H8C,&H1C                          :REM mov [si],DS
9050 DATA &H5E,&H5D                          :REM pop si:pop bp
9060 DATA &HCA,&H02,&H00                     :REM ret 2

9100 REM set up routine
9110 FOR I=1 TO 14
9120   READ CODE%                            :REM get machine code
9130   GETSEG$=GETSEG$+CHR$(CODE%)           :REM store it
9140 NEXT I

9150 GETSEG=PEEK(VARPTR(GETSEG$)+1)+(256*PEEK
     (VARPTR(GETSEG$)+2)

9160 RETURN
```

Notes:

1. To use the above routine do a GOSUB 9100 at the beginning of the program and then call it, for example

   ```
   10 GOSUB 9100
   20 CALL GETSEG(SEG%) ' returns Data Segment in SEG%
   ```

2. Statement 9150 assigns the address of the string GETSEG$ in the variable GETSEG. Strings are usually to be found at the end of the data segments and have offsets greater than 32767. Therefore an integer variable e.g. GETSEG% cannot be used to obtain the address easily.

3. The function VARPTR returns the address of a 3 byte string descriptor of the following format:

   BYTE string length in bytes
   WORD offset to string data (Intel format)

   Statement 9150 picks up the contents of the WORD from the string pointer packet.

Non-BIOS routine calls

Certain calls to installed Device drivers require a Segment:Offset pointer to a parameter block in a register pair.

The routine given below is used to call an external Device driver with a Parameter block in a string variable. The interface extracts the pointer to the parameter block from the string descriptor and places it in the SI register before calling the driver.

On entry the stack pointer is the same as for the example given above.

SP + 4 Address of string descriptor
SP + 2 Segment of Return address
SP + 0 Offset of Return address

Note that the first two Push statements require the offset in the routine to be modified by 4.

```
9200 DATA &H55                 : REM push bp
9201 DATA &H56                 : REM push si
9202 DATA &H8B,&HEC            : REM mov bp,sp
9203 DATA &H8B,&H6E,&H08 : REM mov bp,[bp+8]
9205 DATA &H8B,&H76,&H01 : REM mov si,1[bp]
9206 DATA &HCD,&Hxx           : REM int xxH
                               : where xx is a valid interrupt address
9206 DATA &H5E                 : REM pop si
9207 DATA &H5D                 : REM pop bp
9208 DATA &HCA,&H02,&H00 : REM ret 2

9910 CODE$=SPACE$(255)
9920 CODE=PEEK(VARPTR(CODE$)+1)+(256*PEEK
     (VARPTR(CODE$)+2))
9950 FOR I=0 TO 16:READ J%: POKE VI+I,J%: NEXT I: RESTORE
9990 RETURN
```

Notes:

1. The routine is generated and executed by including the following statements in an Application:

```
10 GOSUB 9910                  'set up routine
20 CALL CODE(PARAMETER$)       ' calls the routine
```

2. The routine at 9910 should be executed as soon as possible in the Application to ensure that CODE$ is not moved around by the interpreter. If there is doubt then, whilst the Application may be slowed down, recreate the CODE$ each time before using it. As will be seen these problems do not occur in compiled languages.

3. Calls from Interpretive Basic are "Far Calls".

Microsoft Compiled Basic

Microsoft Compiled Basic has the same idiosyncrasies as the interpreted version with the exception that string data is not moved around in the same way.

Therefore the main problems to solve are those of obtaining the Data Segment, String pointers and setting up registers correctly.

The two examples given below are equivalent to those given in the interpretive section.

The Data Segment Register

```
                PAGE 60,132
                TITLE  Get the Data Segment register

code    segment byte    public    'code'
        assume  cs:code
        public  getseg  ;make this routine available to
                         basic

getseg  proc    far     ;must be a "Far" routine

        push    bp      ;save current base page register
        push    si      ;save SI register
        mov     bp,sp   ;copy the stack pointer
        mov     si,8[bp] ;load the address return variable
        mov     [si],ds ;store contents of DS in return
                         variable
        pop     si      ;restore the SI register
        pop     bp      ;restore the base page register
        ret     2       ;return to basic

getseg  endp
code    ends
        end
```

Non-BIOS routines

The task of the interface is to set up the DS:SI registers prior to entering the Device driver. The SI pointer is set from the contents of the string descriptor.

```
        PAGE    60,132
        TITLE   Device driver Interface routine
;
CODE    SEGMENT BYTE    PUBLIC  'CODE'
        ASSUME  CS:CODE
        PUBLIC  IFACE
IFACE   PROC    NEAR
;This routine will be called with the following
;stack contents.
;               STACK: param. block offset : +2
;              (SP) ->: return address     :  0
        PUSH    BP              ;save working
                                 registers
        PUSH    SI
        MOV     BP,SP           ;point to stack.
                                 (param.block offset
                                ;will now be 6 bytes
                                 down the stack)
        MOV     BP,6[BP]        ;BP points to string
                                 descriptor
        MOV     SI,1[BP]        ;DS:SI now point to
                                 paramater string
        INT     0xxh            ;call the device at
                                 interrupt xxH
        POP     SI              ;restore the working
                                 registers
        POP     BP
        RET     2
IFACE   ENDP
CODE    ENDS
        END
```

Microsoft 'C' Programming Language

Unlike interpretive and Compiled Basic, the 'C' programming language provides pointers to Parameter blocks in the correct form and also offers a choice of Near or Far calls.

Interfaces are still required as for other languages to obtain the Data Segment to support calls to the BIOS.

Examples of the interfaces are given below. They should be assembled and linked to the Application.

The Data Segment Register

```
        PAGE 60,132
        TITLE   Get the Data Segment register

pgroup  group   code
code    segment byte    public    'code'
        assume  cs:pgroup
        public  getseg          ;make this routine available
                                 to basic

getseg  proc    far             ;routine may be "Far or Near"
        push    bp              ;save current base page register
        push    si              ;save SI register
        mov     bp,sp           ;copy the stack pointer
        mov     si,8[bp]        ;load the address return
                                 variable
        mov     [si],ds         ;store contents of DS in
                                 return variable
        pop     si              ;restore the SI register
        pop     bp              ;restore the base page register
        ret     2               ;

getseg  endp
code    ends
end
```

```
        PAGE    60,132
        TITLE   Device Driver Interface Program
;
PGROUP  GROUP   PROG
PROG    SEGMENT BYTE    PUBLIC  'PROG'
        ASSUME  CS:PGROUP

        PUBLIC  IFACE
IFACE   PROC    NEAR
;This routine will be called with the following
;stack contents:
;           STACK: param. block offset : +2
;           (SP) ->: return address    :  0
        PUSH    BP              ;save working
                                 registers
        PUSH    SI
        MOV     BP,SP           ;point to stack.
                                 (param.block offset
                                ;will now be 6 bytes
                                 down the stack)
        MOV     SI,6[BP]        ;DS:SI now point to
                                 paramater block
        INT     0xxh            ;call the device driver
                                 with interrupt xxH
        POP     SI              ;restore the working
                                 registers
        POP     BP
        RET     2
IFACE   ENDP
PROG    ENDS
        END
```

Index

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Addenda